UNIVERSITÀ
POLITECNICA
DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Meccanica

Modellazione Cinematica con Machine Learning in Matlab

Kinematic Modeling with Machine Learning in Matlab

Relatore:

Prof. Matteo Claudio Palpacelli

Tesi di laurea di:

Savoia Davide

Anno Accademico 2022 / 2023

Indice

Capitolo 1

Introduzione

Il machine learning (ML) è un ramo dell'intelligenza artificiale che si concentra sulla creazione di sistemi capaci di apprendere dai dati, identificare pattern e prendere decisioni con un intervento umano minimo. È basato sul principio che le macchine possano ricevere dati e utilizzarli per auto-migliorarsi attraverso l'apprendimento. Le reti neurali si sono dimostrate strumenti versatili, applicabili a un'ampia gamma di compiti di ML, come il riconoscimento di pattern, la classificazione di immagini, la previsione di serie temporali e il processamento del linguaggio naturale. Le applicazioni del machine learning sono varie e vanno dall'analisi predittiva in finanza, al riconoscimento di immagini in medicina fino al campo meccanico di nostro interesse. Per quanto riguarda invece il deep learning, ciò che possiamo dire è che è una sottocategoria del machine learning che utilizza reti neurali con molteplici strati, o "deep architectures", per elaborare dati con una struttura gerarchica complessa. È particolarmente utile in compiti di percezione, come il riconoscimento visivo e il riconoscimento vocale. In questi casi, il modello deve imparare a identificare pattern complessi e ad astrarsi da essi per fare previsioni o classificazioni accurate È uno strumento potente che ha rivoluzionato il campo dell'intelligenza artificiale, offrendo soluzioni a problemi che un tempo erano considerati inaccessibili alle macchine.

Per questa tesi vedremo l’applicazione del machine learning (mediante reti neurali) in Matlab per il riconoscimento dell’angolo di uscita di un quadrilatero articolato conoscendone la struttura.

1.1 Reti neurali nel campo dell’apprendimento automatico

Le reti neurali, che sono modelli computazionali ispirati al funzionamento del cervello umano, sono uno degli strumenti più potenti a disposizione del machine learning. Esse consistono di nodi, o "neuroni", organizzati in strati, che processano i dati in modo sequenziale. Ogni neurone può trasmettere segnali ad altri neuroni, simulando così il modo in cui i neuroni biologici comunicano tra loro. La rete "apprende" regolando il peso delle connessioni tra i neuroni in base ai dati di input durante il processo di addestramento. L'applicazione delle reti neurali nel machine learning è particolarmente efficace nei compiti di classificazione e regressione.

1.2 Obiettivi e scopo della tesi

L’idea di base di questa tesi è quella di spiegare e rendere il più chiaro possibile il funzionamento delle reti neurali e come queste vengano sfruttate all’interno dell’ambiente di sviluppo Matlab. In particolar modo, vedremo l’applicazione di una rete neurale per l’ottenimento dell’angolo di uscita di un quadrilatero articolato conosciuta la sua struttura. Oltre al caso specifico che tratteremo, sarà importante anche la comprensione di ciò che il software può permettere di ottenere a seconda delle necessità dato che al suo interno presenta molta varietà nell’utilizzo.

1.3 Struttura della tesi

Per esprimere al meglio tutti gli aspetti che verranno trattati, la tesi sarà suddivisa nel seguente modo:

- **Capitolo 2:** Presenterà nel dettaglio le tecnologie utilizzate per lo sviluppo del progetto.
- **Capitolo 3:** Riguarderà la trattazione relativa alle reti neurali artificiali.
- **Capitolo 4:** Tratteremo la struttura di rete neurale di nostro interesse entrando nel dettaglio di tutte le sue caratteristiche
- **Capitolo 5:** Illustrerà nel dettaglio ciò che è stato fatto in Matlab.
- **Capitolo 6:** Conclusioni e sviluppi futuri.

Capitolo 2

Strumenti e metodi

Il progetto è stato sviluppato e scritto interamente in MATLAB. Sia per quel che riguarda la realizzazione dei codici per la creazione della rete neurale ed il suo addestramento, sia per la creazione dei dataset utilizzati per l'addestramento.

2.1 Quadrilatero articolato

La scelta per lo svolgimento di questo elaborato è ricaduta sul quadrilatero articolato (Fig.1). Esso è una configurazione meccanica fondamentale nell'ingegneria, essenziale per il trasferimento controllato del movimento e delle forze all'interno di una struttura o un meccanismo. La sua semplicità di costruzione, unita alla capacità di fornire una vasta gamma di movimenti, lo rende oggetto di studio di base in ogni corso meccanico. Il quadrilatero articolato trova impiego in una molteplicità di contesti per la sua capacità di gestire movimenti dettagliati e regolati.

Nell'ambito dell'ingegneria meccanica, è una scelta comune per la realizzazione di sistemi di sospensione, dispositivi di sterzo e meccanismi di accoppiamento. All'interno del design industriale, questo meccanismo è essenziale in macchinari che necessitano di una gestione accurata del movimento, come le stampanti e i sistemi di taglio. La robotica sfrutta anche il quadrilatero articolato per produrre articolazioni e prensili che replicano le funzioni motorie tipiche degli organismi biologici.

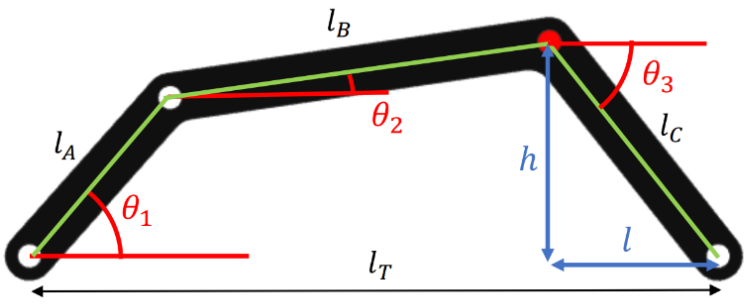


Figura 1 Schematizzazione quadrilatero articolato con indicati i parametri di riferimento

Il meccanismo è composto da quattro aste rigide: due aste di lunghezza fissa, note come bielle, e due aste di lunghezza variabile, chiamate manovelle o bielle motrici, se una delle aste viene utilizzata per guidare il meccanismo. Queste aste sono collegate tra loro da coppie rotoidali che vanno a formare una maglia chiusa. Questa disposizione permette alle aste di muoversi in maniera controllata, trasformando l'input di movimento in un output desiderato, che può essere di natura lineare, rotativa o una combinazione di entrambi.

2.2 Dataset

Il dataset per questo progetto è stato creato e sviluppato direttamente in MATLAB. Si tratta nel particolare di 2 matrici n*1, dove n indica il numero di valori a seconda del numero di dati che volgiamo utilizzare. Una matrice comprendente gli angoli di ingresso del nostro quadrilatero articolato nel range e con lo step di interesse. La seconda al suo interno presenta gli angoli di uscita rispetto all'angolo di ingresso.

2.3 MATLAB

MATLAB si afferma come un linguaggio di programmazione robusto, ampiamente adottato per sviluppare algoritmi complessi in ambiti matematici e scientifici, e si ritrova in uso in una vasta gamma di enti aziendali e istituzionali. Questo linguaggio è accessibile tramite un'applicazione desktop che include una varietà di strumenti utili per settori che vanno dall'automazione industriale fino al settore emergente dell'intelligenza artificiale. Nell'ambito di questa tesi, è stata impiegata la versione R2022a di MATLAB, fornita sotto una licenza accademica concessa dall'Università Politecnica delle Marche.

Capitolo 3

Reti neurali artificiali

Le reti neurali artificiali (ANN) sono il fulcro di molte applicazioni moderne di intelligenza artificiale e machine learning. Le ANN si ispirano, come dice il nome stesso, alla biologia del cervello umano. Queste reti sono formate da una moltitudine di unità di calcolo interconnesse che operano simultaneamente per affrontare compiti specifici. Esse apprendono attraverso l'esperienza anziché tramite istruzioni dettagliate, permettendo loro di riconoscere schemi e tendenze complesse all'interno di grandi insiemi di dati. Un elemento chiave di quest'ultime è la loro abilità nell'auto-perfezionamento progressivo, il che le rende strumenti potenti per attività che sfidano la definizione di regole fisse, come il riconoscimento visivo, l'interpretazione del linguaggio naturale e sistemi di navigazione autonomia. In questo capitolo vedremo le analogie che sono presenti con le reti biologiche del cervello, il loro funzionamento e l'evoluzione che hanno avuto nel corso della storia in modo da far comprendere nella maniera più lineare e completa possibile il loro reale potenziale.

3.1 Rete neurale biologica

I neuroni sono delle cellule elettricamente attive presenti nel cervello umano in grandi quantità, stimato intorno a 10^11. Sebbene ci siano diverse forme di neuroni, la maggior parte di essi ha una struttura simile a quella mostrata nella figura 2.

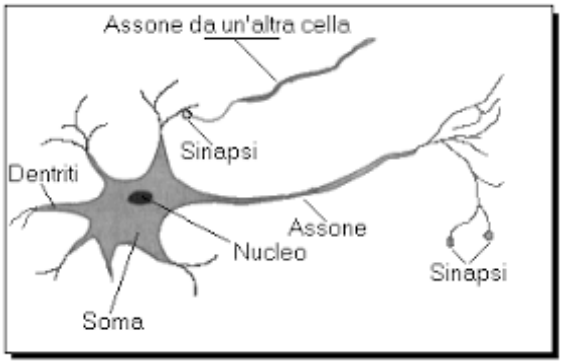


Figura 2 Rete neurale biologica

I neuroni ricevono segnali attraverso le loro strutture di input chiamate dendriti e trasmettono segnali attraverso un'uscita nota come assone. La comunicazione tra i neuroni avviene tramite giunzioni chiamate sinapsi. Ogni neurone è tipicamente collegato a migliaia di altri neuroni andando a formare la tipica rete neuronale (Esempio esplicativo di ciò che intendiamo in figura 3).

Figura 3 Illustrazione rete neurale celebrale

Vediamo nel dettaglio la struttura del Neurone:

1. **Soma (Corpo Cellulare):** Il soma rappresenta il nucleo centrale del neurone, alloggiando il suo DNA e responsabile della gestione delle funzioni vitali della cellula. Inoltre, integra i segnali in ingresso provenienti dai dendriti.
2. **Dendriti:** I dendriti costituiscono prolungamenti ramificati che si estendono dal soma e sono progettati per ricevere segnali sia chimici che elettrici da altri neuroni. Le spine dendritiche, spesso presenti sulla loro superficie, aumentano la superficie disponibile per la formazione di sinapsi con altri neuroni.
3. **Assone:** L'assone è una lunga estensione filiforme del neurone che trasporta l'impulso elettrico dal corpo cellulare ad altre cellule, come neuroni o muscoli. Negli assoni, può manifestarsi un'estensione significativa, raggiungendo persino diversi metri in alcuni neuroni periferici.
4. **Terminali Assonali:** All'estremità dell'assone, troviamo i terminali assonali o bottoni sinaptici, responsabili del rilascio di neurotrasmettitori per comunicare con i dendriti di altri neuroni presso le sinapsi.

I neuroni possono trovarsi in uno dei due stati principali: attivo o a riposo. All'arrivo dei vari input, il corpo cellulare fa una somma pesata di questi segnali in ingresso e, se il risultato supera un certo valore di soglia, il neurone si attiva e produce un "potenziale di azione" che viene inviato all'assone. In alternativa, il neurone rimane in uno stato di riposo. Quando il segnale raggiunge una sinapsi, provoca il rilascio di sostanze chimiche chiamate neurotrasmettitori, che attraversano la giunzione e influenzano l'attività dei neuroni successivi. Questo effetto può essere eccitatorio (aumentando la probabilità che il neurone successivo si attivi) o inibitorio (riducendo tale probabilità), a seconda del tipo di sinapsi. Ogni sinapsi ha un peso associato che determina la sua forza e il suo impatto. Sebbene ogni neurone operi a una scala temporale di millisecondi, il potere computazionale totale della rete neurale è elevato grazie al gran numero di neuroni e sinapsi che possono operare in modo parallelo e simultaneo.

Una caratteristica saliente di queste reti è la loro capacità di modificare la propria architettura in risposta a stimoli esterni, un processo che può essere descritto come "apprendimento", un tratto che i modelli artificiali aspirano ad emulare. Inoltre, la rete neurale biologica mostra una notevole capacità di tollerare informazioni poco precise o errate, è in grado di apprendere e di generalizzare da esperienze passate.

3.2 Neurone di McCulloch-Pitts

3.2.1 Modello del neurone

Nell'ambito della scienza dei computer e delle reti neurali artificiali, il modello neuronale ideato da Warren McCulloch e Walter Pitts (1943) è considerato una colonna portante. Anche se sono emersi modelli più evoluti e sofisticati, il neurone di McCulloch-Pitts (MCP) continua a essere una risorsa essenziale per capire i principi fondamentali alla base delle reti neurali artificiali. Il neurone MCP riduce a un modello basilare il neurone biologico, come si può notare in figura 4.

McCulloch e Pitts teorizzarono che una rete di tali neuroni potesse eseguire qualsiasi funzione computazionale, risultando quindi universalmente capace dal punto di vista computazionale.

Sostenevano che processi mentali complessi, come il ragionamento umano e i meccanismi decisionali, potessero essere ricondotti a operazioni binarie di logica, eseguibili da reti di questi neuroni sintetici. Questo modello, per quanto essenziale, ha avuto un'influenza decisiva nello sviluppo delle teorie dell'informatica e dell'intelligenza artificiale, illustrando come anche gli algoritmi più complessi possano essere interpretati attraverso l'applicazione di regole matematiche e logiche fondamentali.

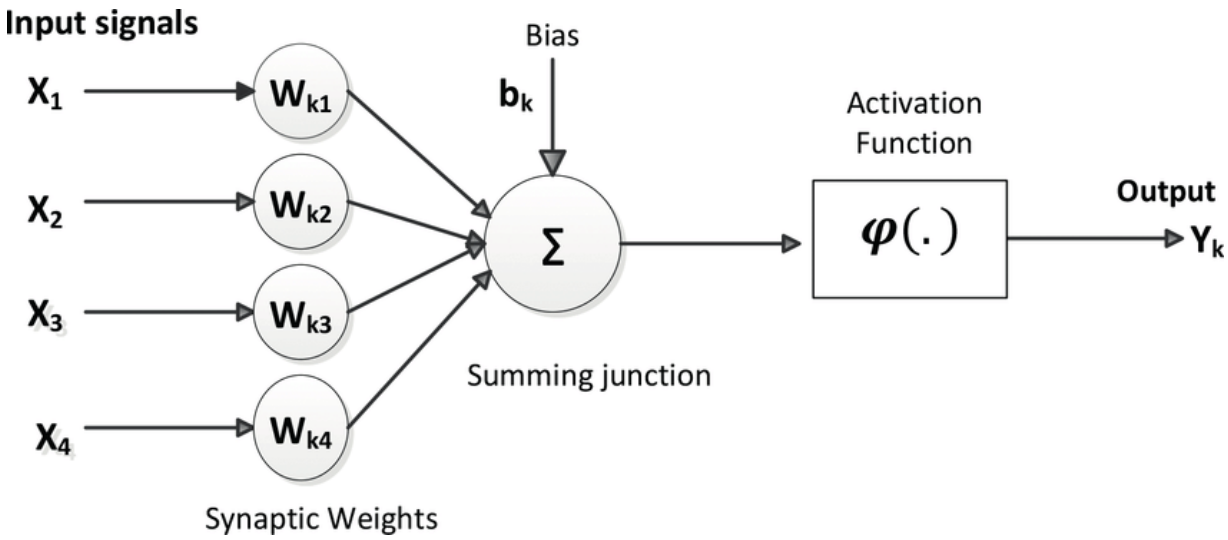


Figura 4 Modello del neurone di McCulloch-Pitts

Presentato come una cellula computazionale binaria, il neurone MCP si articola in tre componenti principali:

1. **Ingressi**: Paragonabili ai dendriti di un neurone biologico, gli ingressi nel neurone di McCulloch-Pitts accettano valori di 0 o 1, indicando la mancanza o la presenza di un segnale.
2. **Sommatore:** I segnali in entrata convergono in un aggregatore che calcola la somma dei valori ponderati per il loro peso sinaptico, che simboleggiano le sinapsi di un neurone naturale.
3. **Funzione di attivazione**: Quando la somma dei segnali ponderati eccede una certa soglia, l'attivatore produce un segnale binario di uscita, ovvero un "potenziale d'azione", che nel caso del neurone MCP è un valore di 0 o 1.

L'impatto di un segnale x su un neurone successivo equivale al risultato della moltiplicazione w • x, dove w rappresenta il peso assegnato alla sinapsi in questione. L'input netto o il potenziale di attivazione A, di un neurone i-esimo, è quindi la somma algebrica dei prodotti fra tutti i segnali di ingresso x_i e i pesi delle sinapsi corrispondenti $w_{i,j}$ a cui va aggiunto il valore del bias b, che ricopre approssimativamente il ruolo di una constante C nelle funzioni, ossia scalai valori di input e determina la facilità con cui l'unità produce un output di 1

(corrisponde alla soglia di attivazione del neurone biologico) e, in certi casi, viene inserita come prima componente dell'ingresso (in quel caso la sommatoria parte da 0 e non più da 1):

$$A_i = \sum w_j \bullet x_i + b_k$$

McCulloch e Pitts teorizzarono che una rete di tali neuroni potesse eseguire qualsiasi funzione computazionale, risultando quindi universalmente capace dal punto di vista computazionale. Sostenevano che processi mentali complessi, come il ragionamento umano e i meccanismi decisionali, potessero essere ricondotti a operazioni binarie di logica, eseguibili da reti di questi neuroni sintetici. Questo modello, per quanto essenziale, ha avuto un'influenza decisiva nello sviluppo delle teorie dell'informatica e dell'intelligenza artificiale, illustrando come anche gli algoritmi più complessi possano essere interpretati attraverso l'applicazione di regole matematiche e logiche fondamentali.

3.2.2 Funzioni di attivazione

La funzione di attivazione, indicata con φ (A), determina l'uscita di un neurone basandosi sull'intensità di attivazione degli input. Essa può assumere valori nell' intervallo [0,1] o [-1,1]. Questo si allinea con la prassi di normalizzare gli input alla rete per assicurare che i segnali che si propagano nella rete rimangano contenuti e sotto controllo. Dunque, avremo che l'output y_k sarà dato da: $y_k = \varphi$ (A).

Esistono diverse funzioni di attivazione, vediamo le più comuni:

- **Funzione di attivazione a gradino**(fig.5): conosciuta anche come funzione di soglia, è la funzione di attivazione usata nel modello di McCulloch-Pitts e rappresenta una forma elementare di attivazione nei

sistemi di reti neurali artificiali. Questa funzione genera unicamente due output possibili, tipicamente valori 0 o 1, secondo la regola:

$$\varphi(A) = \{1\ se\ A \geq 0;\ 0\ se\ A \leq 0$$

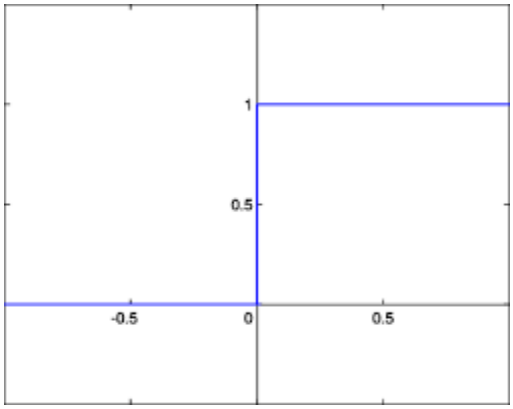


Figura 5 Funzione di attivazione a gradino

Questa funzione si comporta simile a un commutatore binario: se l'input al neurone eccede un valore soglia specificato, il neurone si attiverà producendo un output di 1; se l'input è inferiore, il neurone rimarrà inattivo con un output di 0. Ad oggi è caduta in disuso nelle architetture neurali perché la sua indisponibilità alla differenziazione impedisce l'applicazione della discesa del gradiente durante la fase di apprendimento.

- **Funzione di attivazione ReLu**(rectified linear unit)(fig.6): è una delle funzioni di attivazione più comunemente utilizzate nelle reti neurali, specialmente nelle reti neurali profonde. Si tratta di una funzione di attivazione non lineare e ha la seguente espressione:

$$\varphi(A) = \max(0, A)$$

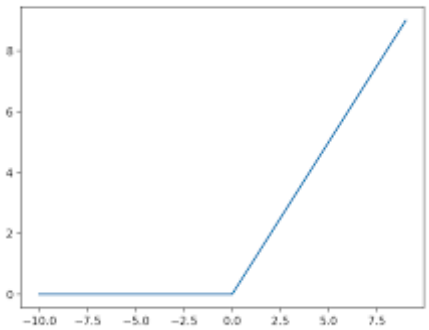


Figura 6 Funzione di attivazione ReLu

Ciò significa che se l'input A *è positivo, la funzione restituirà A*; se A è negativo, restituirà 0. Possiamo individuare diversi aspetti positivi nell'utilizzo di questa funzione di attivazione, come, per esempio:

- Comportamento lineare per input positivi: Questa caratteristica aiuta a ridurre il fenomeno del gradiente sparito, tipico di altre funzioni di attivazione come la sigmoide o, che si manifesta frequentemente nelle reti neurali con molti strati.
- Semplicità di calcolo: La ReLU è più facile e veloce da computare in confronto ad altre funzioni di attivazione complesse.
- Attivazione sparsa: Poiché la ReLU assegna il valore zero a qualsiasi input negativo, ciò porta a un'attivazione sparsa che può essere vantaggiosa in termini di efficienza di calcolo e capacità di rappresentazione dei dati.

Ciononostante, la ReLU non è esente da svantaggi, tra cui il problema dei "neuroni inerti", che si verifica quando i neuroni cessano di attivarsi a seguito di aggiornamenti dei pesi che impediscono loro di fornire output positivi. Questo problema può essere ovviato sfruttando, per esempio, la funzione di attivazione lineare a tratti, le Leaky ReLu e la Parametric ReLu.

- **Funzione di attivazione sigmoide**: Funzione di attivazione non-lineare tra le più usate presenta la seguente espressione:

$$\varphi(A) = 1 / (1+e^{-A})$$

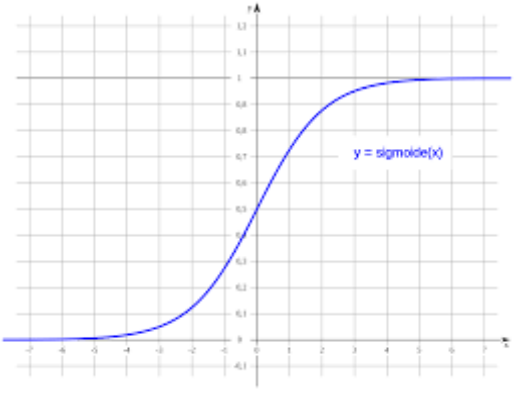


Figura 7 Funzione di attivazione sigmoide

La curva della sigmoide ha una forma a "S" e tende ad avvicinarsi a 1 per valori di input molto grandi in positivo e a 0 per valori di input molto grandi in negativo, con una transizione relativamente graduale in prossimità dell'origine. La funzione sigmoide è una scelta popolare per le reti neurali perché è differenziabile e la sua derivata è facile da calcolare, il che è utile per l'algoritmo di backpropagation, un meccanismo di aggiornamento dei pesi nella formazione delle reti neurali (verrà trattato in maniera più approfondita in futuro) durante l'addestramento della rete.

Tuttavia, ha anche degli svantaggi come il fenomeno del “gradiente che svanisce”, il quale consiste, come consiglia il nome, nella scomparsa del gradiente o, più precisamente, nel suo tendere asintoticamente a zero. Questa riduzione è spesso osservata in reti che utilizzano la funzione di attivazione sigmoide o funzioni simili che possono saturare. Se il gradiente si riduce troppo, l'effetto degli aggiornamenti di peso negli strati vicino all'input diventa trascurabile.

Di conseguenza, questi strati apprendono a un ritmo estremamente lento o possono addirittura smettere di apprendere. Questo porta a un rallentamento del processo di apprendimento della rete o può causare che la rete si fermi su livelli di performance non ottimali prima di aver completamente appreso i pattern desiderati. Per ovviare a questo problema si possono utilizzare funzioni di attivazione, come ad esempio la ReLu (rectified linear unit) vista precedentemente, che non saturano facilmente

- **Funzione di attivazione tansigmoide** : La funzione di attivazione tansigmoide(fig.8), nota anche come funzione tangente iperbolica o tanh, è una funzione matematica comunemente usata nelle reti neurali artificiali, in particolare in quelle con architetture di tipo feedforward e backpropagation. La sua espressione matematica è la medesima:

$$\tanh(A) = (e^{A} - e^{-A}) / (e^{A} + e^{-A})$$

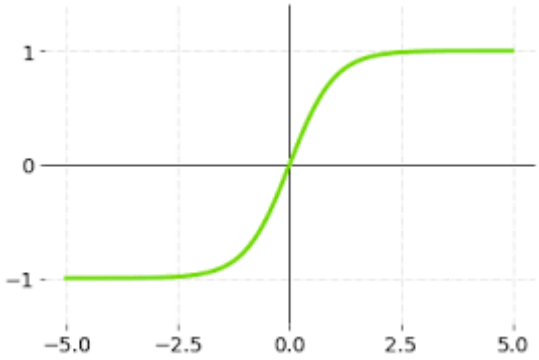


Figura 8 Funzione di attivazione tanh

Una delle principali caratteristiche della funzione tanh è la sua centratura zero. Questo significa che la funzione mappa i valori di input in un intervallo che va da -1 a +1. Questa proprietà è particolarmente vantaggiosa durante il processo di backpropagation, in quanto permette ai gradienti di fluire meglio nella rete, favorendo un apprendimento più equilibrato e veloce. La centratura zero aiuta a prevenire i problemi legati ai pesi, che possono verificarsi quando i valori sono tutti positivi (come nella funzione sigmoide). Un altro vantaggio significativo della tanh è la sua capacità di mitigare il problema del vanishing gradient. Questo problema si verifica quando i gradienti diventano troppo piccoli per apportare modifiche significative ai pesi durante il training, rallentando o bloccando completamente l'apprendimento. Grazie alla sua forma, la tanh è meno soggetta a questo problema, soprattutto per valori di input vicini allo zero, rendendola più efficace della sigmoide in molti scenari. Per questo motivo è la funzione di attivazione che utilizzeremo, poiché rispecchia tutte quelle caratteristiche che cerchiamo per la nostra applicazione.

3.3 Tipologie di reti neurali

Esiste una esorbitante varietà di architetture di reti neurali che possiamo utilizzare a seconda della tipologia del problema da risolvere, dimensione dei dati, capacità computazionale, complessità del modello e altre caratteristiche ancora. Tutta via, si può fare una distinzione fra le 2 famiglie fondamentali da cui discendono poi tutte le varie architetture, ossia le Feedforward e le Feedback o ricorsive.

3.3.1 Reti feedforward

Le reti neurali feedforward sono la forma più diretta e semplice di rete neurale. In queste reti, l'informazione si muove in una sola direzione: dall'input all'output, senza cicli né percorsi di ritorno. L'esempio più basilare di rete feedforward è il percettrone(fig.9), che consiste in un unico strato di neuroni di output denominato Single-Layer Perceptron.

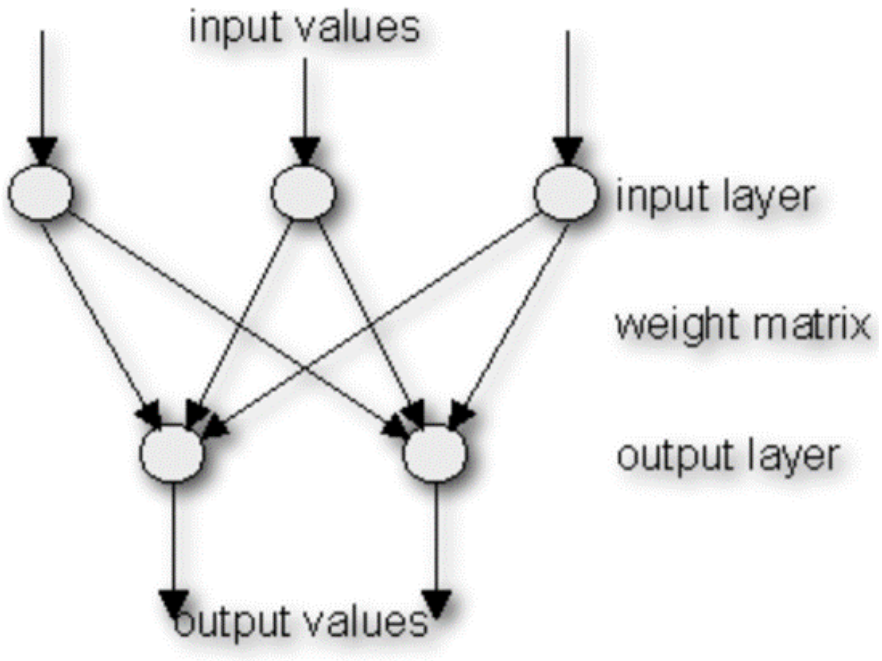


Figura 9 Single-Layer-Perceptron

Tuttavia, la maggior parte delle reti feedforward moderne possiede più strati nascosti, creando un modello noto come Multi-Layer Perceptron (MLP)(fig.10).

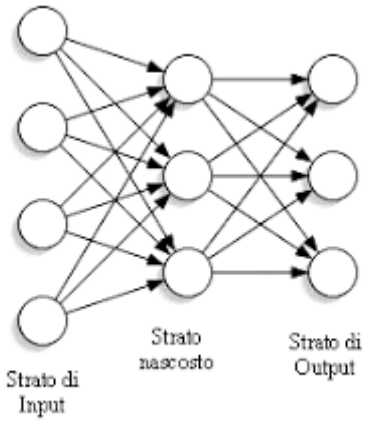


Figura 10 Multi-Layer-Perceptron

Ogni neurone in uno strato feedforward è connesso a ogni neurone nello strato successivo attraverso dei pesi che rappresentano la forza delle connessioni sinaptiche nel cervello. Durante la fase di addestramento, questi pesi vengono aggiustati tramite algoritmi come la discesa gradiente, per minimizzare la differenza tra l'output della rete e l'output desiderato. Come applicazione le troviamo nel caso di classificazione, regressione, riconoscimento di pattern, e altre attività di previsione dove il contesto temporale non è rilevante.

Vediamo ora una successione di quali sono le strutture neurali più conosciute e utilizzate all'interno di questa famiglia:

1. **Multi-layer perceptron (MLP)**: Il MLP è particolarmente noto per la sua applicazione nella classificazione, dove i dati vengono assegnati a categorie discrete, e nella regressione, dove si prevedono valori continui. Su questa tipologia di rete è ricaduta la scelta per il nostro progetto, dunque, gli sarà dedicato più avanti un capitolo per la sua trattazione in modo più approfondito e accurato, entrando nei dettagli del funzionamento, delle possibili problematiche e accorgimenti da avere.

2. **Reti neurali convoluzionali (CNN)**: Le CNN hanno trasformato il campo della visione artificiale, emergendo come uno degli strumenti più potenti per il riconoscimento di immagini e video. Queste reti specializzate

simulano il modo in cui il sistema visivo umano interpreta le immagini, attraverso un processo che emula il riconoscimento di pattern visivi. Sfruttano l'architettura convoluzionale per gestire efficientemente i dati immagine ad alta dimensionalità. Queste reti sono costituite da diversi strati che imitano il processo di visione umana:

-Strati Convoluzionali: Il cuore delle CNN, questi strati utilizzano filtri che scorrono sull'immagine per catturare le caratteristiche locali, come bordi e angoli.

-Strati di Attivazione: Tipicamente, dopo ogni convoluzione, segue una funzione di attivazione non lineare come ReLU, che introduce non linearità nel sistema permettendo alla rete di apprendere complessi pattern visivi.

-Strati di Pooling: Questi strati riducono la dimensione delle caratteristiche preservando le più importanti, rendendo la rete più efficiente e meno incline all'ove fitting.

-Strati Fully-Connected (Dense): Dopo i vari strati convoluzionali e di pooling, la rete utilizza uno o più strati fully-connected per classificare le immagini in base alle caratteristiche estratte.

3. **Autoencoders (AE)**: Gli auto encoder sono una classe di algoritmi di deep learning che servono a creare rappresentazioni compatte di dati complessi in modo non supervisionato. La loro struttura è composta da tre parti principali: l'encoder, il codice e il decoder. L'encoder comprime i dati, mentre il decoder li ricostruisce. Questi modelli vengono addestrati per minimizzare l'errore di ricostruzione dei dati, imparando così le caratteristiche fondamentali di questi ultimi. Sono usati in vari ambiti, come la riduzione della dimensionalità, il rilevamento di anomalie, la rimozione di rumore dai dati e la generazione di dati sintetici.

3.3.2 Reti Neurali Feedback o Ricorsive

Le reti neurali feedback (fig.11), anche note come reti neurali ricorrenti (RNN), differiscono significativamente dalle reti feedforward. In queste architetture, l'informazione può viaggiare in entrambe le direzioni grazie alla presenza di cicli nella rete. Questo permette alla rete di avere una forma di memoria interna, mantenendo traccia delle informazioni passate per influenzare i calcoli futuri. Le RNN sono strutturate in modo tale che la loro uscita ad un certo tempo t dipenda non solo dall'input corrente ma anche dai propri stati precedenti. Questa caratteristica le rende particolarmente adatte per applicazioni come il riconoscimento della voce, la traduzione automatica, e la generazione di testo, dove il contesto e la sequenzialità sono importanti. Nonostante la loro potenza, le RNN tradizionali soffrono di alcuni problemi come la difficoltà nel catturare dipendenze a lungo termine a causa di problemi noti come "scomparsa del gradiente". Per superare questi limiti, sono stati introdotti modelli più avanzati come le Long Short-Term Memory (LSTM) e le Gated Recurrent Unit (GRU), che possono mantenere l'informazione per periodi più lunghi e sono più efficaci nell'apprendere sequenze di dati.

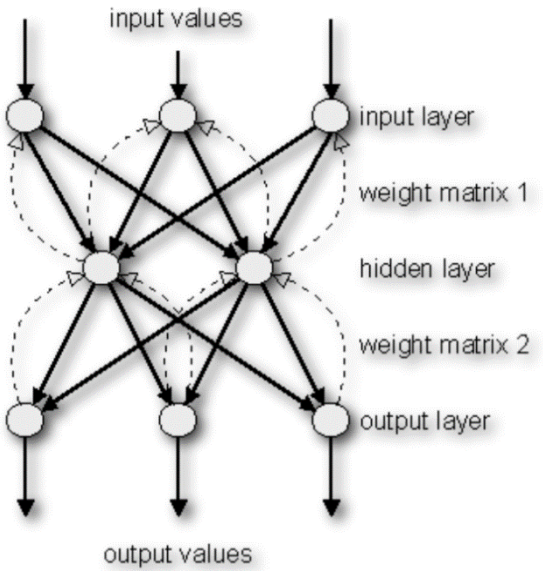


Figura 11 Reti neurali feedback

Anche per questa categoria vediamo quali sono le strutture più utilizzate:

1. **Reti Neurali Ricorrenti** (RNN): Sono particolari tipologie di reti neurali che si caratterizzano per la loro capacità di mantenere una memoria degli ingressi passati grazie alla presenza di cicli nella loro architettura. Ciò consente di gestire dati sequenziali e dipendenti temporalmente, come il linguaggio naturale o le serie temporali. Tuttavia, le RNN tradizionali sono soggette a problemi di vanishing o exploding gradient durante la fase di addestramento.
2. **Long Short-Term Memory** (LSTM): Queste reti sono una tipologia avanzata di reti neurali ricorrenti (RNN) progettate per superare il problema del vanishing gradient riscontrato nelle RNN tradizionali. Gli LSTM sono in grado di imparare dipendenze a lungo termine e sono meno sensibili alla lunghezza degli intervalli temporali, il che li rende superiori ad altri metodi di apprendimento sequenziale in compiti come la classificazione, l'elaborazione e la previsione di dati basati su serie temporali. Un'unità LSTM è composta da una cella e tre gate: il gate di ingresso, il gate di uscita e il gate di dimenticanza, che regolano il flusso delle informazioni. Queste strutture permettono agli LSTM di ricordare e dimenticare selettivamente le informazioni, che è fondamentale per elaborare sequenze di dati dove la rilevanza delle informazioni può variare nel tempo.
3. **Gated Recurrent Unit** (GRU): Le GRU sono una variante delle reti RNN per superare il problema del gradiente che svanisce. Esse si avvalgono di due gate principali: l'update gate e il reset gate, che aiutano a determinare quali informazioni passate sono rilevanti e quali possono essere scartate, permettendo così al modello di conservare anche informazioni datate. Le GRU sono meno complesse delle LSTM in quanto dispongono di soli due gate contro i tre delle LSTM, il che le rende meno onerosi in termini di calcolo, particolarmente adatti a serie di dati più piccole. Entrambi i sistemi sono ampiamente utilizzati in compiti di apprendimento automatico complessi, come l'analisi del sentimento, il riconoscimento vocale e la traduzione automatica, diventando strumenti indispensabili nella scienza dei dati.

3.5 Tipologie di apprendimento

Nell'ambito delle reti neurali artificiali, esistono diversi tipi di apprendimento che definiscono come una rete può modificare i propri pesi sinaptici in risposta agli input ricevuti. Il tipo di apprendimento è determinato quindi da come questi adattamenti avvengono. Ecco le tipologie principali:

1.**ApprendimentoSupervisionato:** L'apprendimento supervisionato nelle reti neurali è un metodo in cui la rete è addestrata su un insieme di dati etichettati, ossia, ogni esempio di input nel set di dati è accoppiato con l'output corretto (l'etichetta). L'obiettivo è quello di sviluppare un modello che possa fare previsioni accurate sugli output quando gli vengono forniti nuovi input. Questa tipologia sarà quella utilizzata nel nostro specifico caso.

2. **Apprendimento Non Supervisionato:** L'apprendimento non supervisionato nelle reti neurali si riferisce a un tipo di algoritmi che operano su dati non etichettati, cioè dati che non hanno un'etichetta di output specificata o una classificazione nota. Invece di imparare a predire un'etichetta o un valore, queste reti cercano di scoprire strutture intrinseche nei dati.

3. **Apprendimento per Rinforzo:** L'apprendimento per rinforzo è un tipo di apprendimento in cui un agente impara a prendere decisioni ottimizzando le azioni basate sulle ricompense e le punizioni che riceve. L'agente interagisce con un ambiente, riceve feedback sotto forma di ricompense o punizioni e aggiusta le sue politiche di azione di conseguenza.

4. **Apprendimento Semi-Supervisionato**: L'apprendimento semi-supervisionato rappresenta un approccio misto nell'ambito dell'apprendimento automatico, in cui il sistema viene allenato utilizzando un insieme di dati parzialmente etichettati. Alcuni input del set di allenamento sono accompagnati dalle corrispondenti uscite desiderate, similmente a quanto avviene nell'apprendimento supervisionato, mentre altri input ne sono sprovvisti, come accade per l'apprendimento non supervisionato. Il fine ultimo rimane lo stesso: dedurre regole e funzioni che risolvano problemi specifici e scoprire modelli e configurazioni nei dati che aiutino a conseguire obiettivi prefissati.

Ogni metodo di apprendimento è progettato per affrontare specifiche categorie di problemi, e la selezione della tecnica appropriata si basa sulla natura dei dati a disposizione, sulla tipologia del problema da affrontare e sul livello di efficacia che si intende raggiungere.

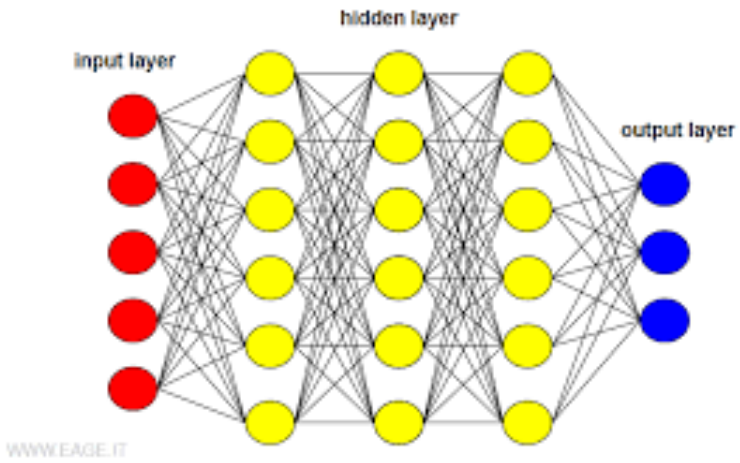
Capitolo 4

Multi-layer perceptron

Per selezionare l'architettura di rete neurale più adatta al nostro scopo, dobbiamo escludere quelle non pertinenti: non utilizziamo immagini e non miriamo a classificare elementi in categorie; quindi, scartiamo le reti neurali convoluzionali (CNN). Poiché la temporalità non è cruciale nel nostro contesto statico, escludiamo anche le reti LSTM, pur riconoscendone il potenziale in applicazioni future, come nella robotica collaborativa. Ciò che cerchiamo è prevedere valori specifici, in un'analisi che può essere assimilata a una regressione. Per questo, abbiamo optato per un Multi-Layer Perceptron (MLP), una rete feedforward che utilizza neuroni interconnessi che attraverso la backpropagation approssimano funzioni continue e affrontano problemi di non linearità.

4.1 Struttura del MLP

Le reti neurali multistrato si compongono di vari layer: uno di ingresso che accoglie i dati, uno o più strati intermedi che trattano questi dati, e uno strato finale che emette il risultato. È possibile vedere in figura 12 una esemplificazione grafica della suddivisione dei 3 layer, input layer, Hidden layer e output layer.



.

Figura 12 Illustrazione rete neurale multistrato con vari strati nascosti

Prima di entrare nel dettaglio di questi 3 strati vediamo da cosa sono composti:

- **Neuroni:** Questi neuroni artificiali ricevono segnali e, a seconda del layer nel quale ci troviamo, riceverà rispettivamente o gli input di ingresso, o input da neuroni dello strato precedente.

- **pesi:** I neuroni interagiscono attraverso collegamenti pesati, parametri che la rete può aggiustare. Questi pesi influenzano la portata del segnale che viene trasmesso da un neurone all'altro. Ciò è presente sono negli strati fi Hidden layer e output layer

- **bias**: Aggiunto a ciascun neurone, il bias è un valore che modifica l'input pesato, permettendo di calibrare la soglia di attivazione del neurone per un'adattabilità ottimale ai dati. Esso è presente nel Hidden layer e nell'output layer.

- **Funzione di attivazione**: Una funzione di attivazione elabora la somma ponderata (composta da una combinazione lineare e un bias). Tale funzione può assumere una forma non lineare, ad esempio la ReLU (Unità Lineare Rettificata), la funzione sigmoide o la tangente iperbolica (Le vedremo più avanti nel dettaglio). L'introduzione di non linearità attraverso la funzione di attivazione consente al modello di imparare e di esprimere correlazioni complesse presenti all'interno dei dati. Esse sono presenti nell'Hidden layer e nell'output layer.

Vediamo ora come si compongono nel dettaglio ogni layer, le loro caratteristiche e le funzioni che svolgono all'interno della rete:

• **Input layer:** L'input layer agisce come ponte tra i dati provenienti dal mondo esterno e il modello di machine learning all'interno della rete. Costituisce il primo stadio di una rete neurale e ha il compito principale di accogliere i dati di partenza destinati alla rete.

Piuttosto che processare o alterare questi dati attraverso calcoli, questo strato agisce come un tramite, trasferendo le informazioni non elaborate agli strati successivi della rete. È composto da neuroni (o nodi) e ciascuno di essi rappresenta una singola caratteristica di input fornita alla rete e il valore non è altro che il valore grezzo della caratteristica di input. Il suo funzionamento si può suddividere in 3 semplici passaggi:

- Ricezione dei dati: Nelle fasi iniziali sia di apprendimento lo strato di input si occupa di accogliere i dati in entrata.
- Standardizzazione dei dati: Per facilitare un apprendimento più efficiente, i dati sono spesso standardizzati prima di procedere oltre, normalizzandoli entro un intervallo standard, come 0 a 1 o -1 a 1 (come visto nelle funzioni di attivazione)
- Inoltro dei dati: Dopo averli accolti, lo strato di input inoltra i dati così come sono agli strati successivi. È soltanto arrivati al primo strato nascosto che i dati subiscono una trasformazione, attraverso l'applicazione di pesi e funzioni di attivazione.

• **Hidden layer:** Gli hidden layers rappresentano il nucleo centrale delle reti neurali artificiali. Ricevono la denominazione di "nascosti" poiché non corrispondono agli strati di input o di output e non interagiscono direttamente né con i dati in ingresso in forma pura né con i risultati finali. Posizionati tra l'input e l'output, questi strati eseguono la parte predominante dei calcoli richiesti per convertire gli ingressi in uscite. In maniera automatica, gli strati nascosti identificano e isolano le caratteristiche salienti all'interno dei dati, che sono fondamentali per eseguire predizioni o classificazioni accurate. Ciascun hidden layer è capace di sviluppare una comprensione dei dati a vari livelli di astrazione. Mentre i livelli più superficiali si focalizzano sull'assimilazione di elementi basilari, gli strati più in profondità si specializzano nell'apprendere dettagli sempre più complessi e astratti. Il numero di questi strati nascosti determina l'effettiva complessità della nostra rete. Ciascun neurone all'interno di questi strati riceve segnali da numerosi neuroni posizionati nello strato che lo precede. I neuroni sono collegati tra loro attraverso pesi, che sono i parametri che la rete può modificare. Questi pesi influenzano l'intensità del segnale trasmesso da un neurone all'altro. Aggiunto a

ciascun neurone, il bias modifica il risultato della loro somma pesata, permettendo così di regolare la soglia della funzione di attivazione per una migliore adattabilità del modello ai dati. Vediamo il suo funzionamento:

- Somma Ponderata: All'interno di ogni neurone, si effettua la somma dei prodotti degli input provenienti dai neuroni dello strato antecedente per i rispettivi pesi. A tale somma si aggiunge il bias.
-Attivazione Neuronale: La somma appena calcolata, inclusivo del bias, viene elaborata mediante una funzione di attivazione, che potrebbe essere di tipo non lineare come ReLU o sigmoide. Queste funzioni inseriscono non linearità nella rete, rendendola capace di decifrare e modellare relazioni complesse all'interno dei dati.
-Trasmissione dell'Output: Dopo che è stata applicata la funzione di attivazione, l'output generato da ogni neurone è trasmesso al livello successivo di neuroni nella rete.

• **Output layer**: L'output layer rappresenta il livello conclusivo di una rete neurale e si occupa di generare l'uscita definitiva per l'attività specifica per cui la rete è stata programmata, che si tratti di classificazione, regressione o produrre altri tipi di risultati a seconda dalla struttura della rete. La composizione è la medesima vista negli strati precedenti ossia, neuroni, pesi, bias e funzione di attivazione in base al tipo di problema da risolvere. Vediamo nel dettaglio il suo funzionamento:

- Produzione dell'Output: Una volta completati i calcoli dagli strati nascosti, l'informazione è inviata all'output layer, che la sintetizza e la converte nel risultato finale emesso dalla rete.
-Determinazione della Classe: Nelle attività di classificazione, è compito dell'output layer stabilire la probabilità delle varie classi basandosi sugli input processati.
-Stima dell'Output: In scenari di regressione, l'output layer restituisce una previsione sotto forma di valore o serie di valori continui.

-Valutazione dell'Errore: In fase di addestramento, si misura l'errore come la discrepanza tra ciò che la rete prevede e il valore obiettivo desiderato, e questo errore è ciò che guida l'aggiornamento dei pesi attraverso la retro-propagazione. In conclusione, lo strato di output è essenziale per decodificare i segnali processati dagli strati intermedi e convertirli in una forma che riflette la scelta conclusiva o l'anticipazione finale effettuata dalla rete

4.2 Applicazioni e prestazioni

Essendo una architettura neurale relativamente semplice rispetto alle altre non è difficile pensare che abbia trovato enorme impiego data la sua semplicità e duttilità. Essa ha è stata impiegato con successo in numerosi settori grazie alla sua efficacia nell'apprendere legami complessi e non lineari presenti nei dati. Esamineremo ora in modo approfondito le sue diverse applicazioni, l'efficacia in vari contesti e, infine, le sfide e i limiti associati a questa tecnologia. Le applicazioni in cui trova impiego sono:

1. **Classificazione**: Data sua struttura flessibile e la sua abilità nell'individuare legami non lineari rendono il Multi-Layer Perceptron (MLP) particolarmente adatto per affrontare compiti di classificazione. In tali compiti, il processo consiste nell'assegnare nuovi input a specifiche categorie o classi basandosi su un set di dati precedentemente addestrato. Durante la fase di addestramento, l'MLP si avvale di algoritmi come la backpropagation e il gradiente discendente per affinare i suoi pesi, mirando a minimizzare la discrepanza tra le previsioni del modello e gli output reali. Prima di essere immessi nel MLP, i dati vengono sottoposti a pre elaborazione. All'interno dell'MLP, ogni neurone processa questi dati e passa il proprio output al livello successivo. Infine, nello strato di output, il modello classifica ogni input in una categoria. In un contesto di classificazione binaria, ciò si traduce nella scelta tra due classi. Per la classificazione multi-classe, invece, la funzione soft Max viene utilizzata per assegnare una probabilità a ogni classe, selezionando poi quella con la probabilità più elevata.

2. **Regressione**: Essendo questa la tecnica che andremo ad adottare nella nostra trattazione gli sarà dedicato un approfondimento maggiore rispetto alle altre. Per iniziare, possiamo dire che anche nel contesto della regressione nell'ambito dell'apprendimento automatico, il MLP si rivela uno strumento molto efficace. A differenza della classificazione, che si occupa della categorizzazione in classi distinte, la regressione si concentra sulla previsione di valori continui. Grazie alla sua abilità nel discernere relazioni complesse e non lineari tra i dati, l'MLP è particolarmente adatto per questo tipo di compito. L'approccio adottato per addestrare un MLP in un contesto di regressione è analogo a quello impiegato per la classificazione. La rete sfrutta algoritmi quali la backpropagation e il gradiente discendente per affinare i suoi pesi, con l'obiettivo di ridurre al minimo l'errore nel modello, spesso misurato tramite l'errore quadratico medio. Dopo l'addestramento, l'MLP diventa capace di generare previsioni di valori continui su nuovi input.

Per quanto riguarda invece gli aspetti più critici, analizziamo dove questa tecnica può andare in difficoltà. Sicuramente determinare la configurazione ottimale di strati e neuroni è cruciale ma non sempre diretto, e richiede sperimentazione. Come nella classificazione, esiste il rischio che l'MLP si adatti troppo ai dati di addestramento, compromettendo la sua capacità di generalizzare a nuovi dati andando così in “over fitting” (Questo fenomeno e la sua risoluzione verranno approfonditi più avanti). Inoltre, la regressione richiede spesso una cura particolare nella preelaborazione dei dati, come la normalizzazione o la standardizzazione, per ottimizzare le prestazioni. Anche in questa tecnica, gli MLP possono mancare di trasparenza nelle loro previsioni, rendendo più complesso comprendere il modo in cui i dati di input influenzano l'output.

3. **Riconoscimento di Pattern**: Il MLP gioca un ruolo cruciale nel riconoscimento di pattern, un settore dell'apprendimento automatico dedicato all'individuazione di schemi ricorrenti all'interno dei dati. Questa capacità dell'MLP di gestire e categorizzare dati non lineari e complessi lo rende particolarmente efficace in questa area. Con la sua abilità di trattare diversi tipi di dati, come segnali audio, video, immagini e testo, l'MLP è in grado di identificare e isolare elementi chiave che sono essenziali nel riconoscimento di pattern. Utilizzando strati nascosti e funzioni di attivazione non lineari, è capace di comprendere e rappresentare le relazioni complesse presenti nei dati, che sono una caratteristica comune nei problemi di riconoscimento di pattern. Durante il processo di addestramento, l'MLP impara attivamente a individuare schemi e tendenze all'interno dei dati di input, regolando i propri pesi e bias per aumentare l'accuratezza sia nella classificazione sia nell'identificazione di specifici pattern.

4. **Elaborazione del Linguaggio Naturale (NLP): Nel** settore dell'Elaborazione del Linguaggio Naturale (NLP), che studia come i computer possano comprendere e interagire con il linguaggio umano, il MLP ha trovato un'efficace area di applicazione. Grazie alla sua abilità nell'interpretare e apprendere da relazioni dati complesse e non lineari, l'MLP si dimostra particolarmente utile per una serie di funzioni nell'NLP. Tra queste funzioni, l'MLP si distingue nella capacità di classificare documenti in diverse categorie, ad esempio separando recensioni positive da quelle negative o organizzando articoli di notizie per argomento. Inoltre, nell'ambito dell'analisi del sentimento, è in grado di discernere e categorizzare le emozioni veicolate in un testo, classificandole come positive, negative o neutre. Anche se gli MLP non rappresentano la tecnologia più avanzata per la generazione di testo, specialmente in confronto a modelli come le reti neurali ricorrenti o i Transformer, hanno comunque la capacità, seppur limitata, di generare testi imparando sequenze e strutture linguistiche dai dati di addestramento. Gli MLP vengono utilizzati anche per sviluppare modelli linguistici di base, apprendendo come le parole sono distribuite e sequenziate in una lingua. Sono capaci di comprendere il linguaggio naturale, estraendo informazioni o rispondendo a domande, e possono

identificare entità nominate come nomi di persone, luoghi o organizzazioni all'interno di testi. Nell'area dei sistemi di raccomandazione, gli MLP possono analizzare descrizioni o recensioni di prodotti per suggerire articoli rilevanti agli utenti. Sebbene non siano la prima scelta per applicazioni di traduzione automatica complessa, gli MLP sono comunque utili per traduzioni più semplici. Per quanto riguarda il riconoscimento vocale, gli MLP possono essere coinvolti nella trasformazione del parlato in testo, anche se per compiti più avanzati si preferiscono generalmente modelli specifici per il riconoscimento vocale.

4.3 Algoritmo di backpropation

L'algoritmo di retro-propagazione dell'errore gioca un ruolo essenziale nel processo formativo delle reti neurali. Questa tecnica è vitale per ottimizzare i parametri della rete basandosi sull'errore rilevato ad ogni ciclo di addestramento, contribuendo così a diminuire la frequenza degli errori e a potenziare l'affidabilità del modello migliorandone l'abilità di generalizzare le informazioni.

Nel corso della storia, la tecnica di backpropagation emerse nel corso degli anni '60 e divenne particolarmente nota verso la fine degli anni '80, in seguito alla pubblicazione dello studio "Learning representations by back-propagating errors" da parte di Rumelhart, Hinton e Williams. Questo algoritmo è stato concepito per individuare e correggere gli errori a partire dai risultati finali, risalendo fino agli input iniziali. Questa metodologia è di fondamentale importanza per affinare le previsioni sia nel campo dell'estrazione dei dati sia nell'apprendimento automatico.

Il processo di backpropagation si dedica al calcolo del gradiente della funzione di errore relativamente ai pesi della rete, operazione che si distingue per la sua elevata efficienza. Questo aspetto permette di impiegare tecniche basate sul gradiente per formare reti neurali a più strati in modo più semplice e diretto, senza la necessità di calcolare il gradiente per ogni singolo peso, il che sarebbe decisamente più dispendioso e meno efficace.

Per riassumere, la retro-propagazione dell'errore è una prassi standard nell'addestramento di reti neurali e si rivela fondamentale per analizzare e comprendere il rapporto tra ingressi e uscite all'interno di una rete. Questo, permette di progettare sistemi di apprendimento avanzato in grado di gestire attività decisamente complesse.

4.4 Over fitting

Un problema comune a quasi tutte le tipologie di reti neurali è il fenomeno del l'over fitting. Anche se un modello viene addestrato con un ampio set di dati, ciò non implica automaticamente che sia più efficace. Infatti, una rete troppo adattata ai dati di input può finire per memorizzare gli output corrispondenti piuttosto che apprendere veri pattern. Il risultato è una rete neurale che risponde efficacemente ai dati su cui è stata addestrata, ma fallisce nel riconoscere le dinamiche sottostanti di fenomeni nuovi, rendendo il modello inadatto a contesti differenti.

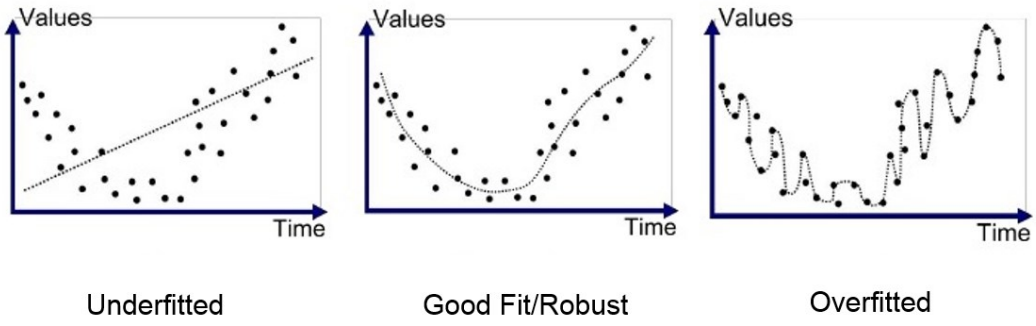


Figura 13 Valutazione grafica del fenomeno dell'over fitting

Per identificare ed evitare l'over fitting, è possibile adottare diverse strategie durante la fase di progettazione della rete.

4.4.1 Suddivisione degli input in 3 set

Una pratica fondamentale è dividere i dati in tre gruppi:

1. **Training set:** esso è il set più grande e contiene le istanze che il modello userà per imparare. Attraverso la ripetizione e l'ottimizzazione, il modello cerca di capire e assimilare i pattern presenti in questi dati. Comprende

input (caratteristiche) e output desiderati (etichette o target). La proporzione tipica del training set rispetto al totale dei dati può variare, ma spesso si attesta intorno al 70%. Durante l'addestramento, il modello effettua iterazioni sui dati del training set, aggiustando i suoi parametri (pesi) per minimizzare l'errore tra le previsioni e i risultati attesi.

2. **Validation set:** Il validation set è un insieme di dati separato, utilizzato per validare la bontà del modello durante il processo di addestramento. Non è utilizzato per l'addestramento diretto, ma per affinare i parametri e la struttura del modello. Solitamente rappresenta circa il 15-20% del dataset totale. I dati in questo set dovrebbero essere scelti in modo casuale ma rappresentativo dell'intero dataset. Usando il validation set, si può monitorare se il modello sta iniziando a soffrire di over fitting. Se l'errore sul validation set inizia ad aumentare mentre quello sul training set diminuisce, è un segnale che il modello si sta adattando troppo ai dati di training. l validation set è utilizzato per regolare gli iperparametri del modello (ad esempio, il tasso di apprendimento, il numero di strati in una rete neurale, ecc.) per migliorarne le prestazioni.

3. **test set:** Il test set è utilizzato per valutare le prestazioni del modello dopo che è stato addestrato e validato. Fornisce una misura imparziale delle prestazioni del modello su dati nuovi e non visti. Comprende dati che non sono stati utilizzati né nell'addestramento né nella validazione. Questo assicura che il test sia una valutazione oggettiva. Le metriche di prestazione calcolate sul test set forniscono un'indicazione su come il modello si comporterà nel mondo reale su dati non visti. È cruciale che il test set sia mantenuto separato e non utilizzato in alcuna fase dell'addestramento per garantire che le valutazioni siano imparziali e rappresentative delle capacità reali del modello.

4.4.2 Convalida incrociata

La validazione incrociata (o cross-validation) è una tecnica cruciale nel processo di valutazione delle reti neurali feedforward, così come in altri tipi di modelli di machine learning. Questa tecnica è particolarmente utile per stimare la capacità di generalizzazione del modello e per prevenire l'over fitting. Questa tecnica è un metodo statistico utilizzato per valutare e migliorare l'efficacia di un modello predittivo, particolarmente utile quando si dispone di un set di dati limitato. Invece di dividere i dati in un singolo training set e validation set fissi, la validazione incrociata ripete questo processo più volte, utilizzando diverse porzioni dei dati come training e validation set in ogni iterazione. Di seguito sono illustrate alcune tipologie di validazione incrociata:

1. **K-Fold Cross-Validation**: La forma più comune di validazione incrociata. I dati sono divisi in "K" gruppi (o "fold") di dimensioni simili. Il modello viene addestrato K volte, ogni volta usando K-1 fold per il training e il fold rimanente per la validazione. Alla fine, si calcola la media delle prestazioni su tutti i K fold per ottenere una stima più robusta dell'efficacia del modello.
2. **Stratified K-Fold Cross-Validation:** Una variazione del K-Fold, utilizzata soprattutto per i dataset con una distribuzione non uniforme delle classi (ad esempio, in problemi di classificazione). In questo caso, ogni fold contiene approssimativamente la stessa percentuale di campioni di ciascuna classe target, come nel set completo.
3. **Leave-One-Out Cross-Validation** (LOOCV): Un caso estremo di K-Fold dove K è uguale al numero di campioni nel set di dati. Ogni singolo dato viene utilizzato una volta come validation set, mentre i restanti servono per il training. Questo metodo è computazionalmente costoso, ma può essere utile per set di dati molto piccoli.

Questa tecnica presenta vari vantaggi fornisce un modo per utilizzare efficacemente tutti i dati disponibili per l'addestramento e la validazione, il che è particolarmente prezioso in scenari con set di dati limitati. Aiuta a ridurre il bias nella stima delle prestazioni del modello, fornendo una visione più accurata di come il modello si comporterà su dati non visti. Inoltre, fornisce informazioni sulla variabilità e la

robustezza del modello. Se le prestazioni del modello variano significativamente attraverso i diversi fold, questo può indicare problemi di generalizzazione. In fine, possiamo dire che è utile per ottimizzare i parametri e la configurazione della rete neurale, permettendo di testare l'efficacia di diverse architetture e parametri di addestramento.

La validazione incrociata può essere computazionalmente costosa, specialmente per reti neurali complesse e grandi set di dati. È importante bilanciare la necessità di una valida stima delle prestazioni con le risorse disponibili. La scelta di "K" in K-Fold cross-validation è critica. Un valore troppo basso potrebbe non fornire una valida stima delle prestazioni, mentre un valore troppo alto potrebbe aumentare il costo computazionale senza benefici significativi. Di estrema importanza è che i dati vengano randomizzati prima di essere divisi in fold per evitare bias dovuti all'ordine dei dati. In conclusione, la validazione incrociata nelle reti neurali feedforward è una strategia potente per valutare e migliorare la qualità dei modelli. Offre un bilanciamento tra la necessità di una stima accurata delle prestazioni e l'utilizzo efficiente dei dati disponibili, riducendo il rischio di over fitting e aumentando la fiducia nelle capacità di generalizzazione del modello.

Per quanto riguarda il nostro specifico caso, questa strategia non è stata implementata poiché il set di dati non rispecchiava la grandezza target per questa tecnica, dunque, sarebbe solo stato uno spreco di tempo e risorse. Ritenevo comunque utile illustrarla poiché, in linea di massima, è estremamente utilizzata.

Capitolo 5

Matlab

Il software utilizzato per questa trattazione è MATLAB (Matrix laboratory)(fig.14) e, come vedremo passo dopo passo in questo capitolo, ha molti strumenti al suo interno che permettono la creazione e sviluppo di reti neurali performanti. MATLAB integra efficacemente calcolo, visualizzazione e programmazione in un ambiente interattivo. Consente di gestire variabili, importare ed esportare dati, effettuare calcoli, creare grafici, scrivere codice e sviluppare applicazioni, rendendolo versatile per molteplici applicazioni

Figura 14 Immagine applicazione Matlab

5.1 Caratteristiche generali

Questo software gestisce grandi set di dati come array e consente calcoli complessi con poche linee di codice. La sua programmazione si caratterizza per l'uso di variabili predefinite, variabili definite dall'utente, vettori, matrici e grafici. Per gestire questa differenziazione di dati, Matlab utilizza tre tipi principali di file:

- **M-file:** Gli M-file hanno estensione ".m" e, siccome sono in formato ASCII (American Standard Code for Information Interchange), è possibile crearli e modificarli tramite qualsiasi editor di testo. MATLAB, però, propone un editor proprio, particolarmente efficace per la redazione di codice MATLAB, grazie alle sue capacità di debug e alla funzione di

evidenziazione sintattica. Utilizzati per automatizzare comandi in MATLAB, gli M-file possono essere semplici script, che eseguono istruzioni in sequenza, o funzioni complesse, capaci di ricevere input, generare output e gestire variabili locali. Essenziali per la programmazione in MATLAB, questi file facilitano la salvaguardia e la riutilizzazione di sequenze di comandi. Il loro principale vantaggio risiede nella possibilità di creare codici sia complessi che riutilizzabili, che possono essere facilmente condivisi e modificati dagli utenti.

- **MAT-file:** I MAT-file, specifici per MATLAB, vengono impiegati per l'archiviazione dei dati. Con un'estensione ".mat", sono formati in binario, il che preclude la loro lettura o modifica tramite editor di testo standard. Il loro scopo principale è conservare le variabili create durante una sessione di lavoro in MATLAB, incluse matrici, vettori, strutture di dati e altri formati gestibili dal software. Questi file sono estremamente utili per salvare i risultati di elaborazioni complesse o ampi insiemi di dati, facilitandone il caricamento in sessioni future senza la necessità di ripetere i calcoli. Il loro vantaggio più significativo risiede nell'efficienza con cui consentono di archiviare e trasferire dati all'interno dell'ambiente MATLAB, assicurando processi di lettura e scrittura dei dati rapidi e agevoli.
- **file dati:** In MATLAB, i file dati sono generalmente rappresentati da file di testo o binari, prodotti da altri software ma compatibili con MATLAB. Essi abbracciano una varietà di formati, inclusi CSV (Comma-Separated Values), TXT (file di testo semplice) e altri formati specifici di programmi diversi. MATLAB è dotato di funzionalità per gestire (leggere e scrivere) questi diversi formati di file. Tali file sono particolarmente utili per l'elaborazione di dati provenienti da fonti esterne a MATLAB, come, ad esempio, un insieme di dati in formato CSV. La capacità di MATLAB di importare ed esportare dati in vari formati lo rende uno strumento efficace per l'analisi di dati da molteplici fonti.

L'interfaccia utente principale di MATLAB si articola attraverso varie finestre interattive, che possono essere organizzate secondo le necessità dell'utente: è possibile posizionarle affiancate, spostarle, minimizzarle o modificarne le dimensioni. Tra queste, le quattro finestre principali, frequentemente utilizzate, sono:

1. **Command Window:** Questa finestra è il cuore dell'interfaccia di MATLAB, dove gli utenti possono inserire comandi direttamente. Qui, i comandi vengono eseguiti, e i risultati vengono visualizzati in tempo reale. È uno strumento fondamentale per l'interazione immediata con MATLAB, permettendo di testare rapidamente codici e funzioni.
2. **Workspace:** La Workspace agisce come un ambiente di archiviazione per le variabili create durante una sessione di MATLAB. Qui, gli utenti possono visualizzare e gestire le variabili attualmente in uso, compresi dettagli come tipo, dimensione e valore delle variabili.
3. **Current Directory:** La finestra Current Directory fornisce una vista interattiva del contenuto delle cartelle sull'hard disk. Permette agli utenti di navigare tra i file e le cartelle, facilitando l'accesso e la gestione dei file di script, funzioni e dati.
4. **Command History:** Nella Command History vengono registrati tutti i comandi recentemente digitati, ordinati per data e ora. Questa funzionalità è particolarmente utile per tracciare le azioni precedenti, ripetere comandi passati o correggere errori senza dover ridigitare l'intero comando.

L'interfaccia di MATLAB, con queste finestre ben organizzate, offre un ambiente di lavoro altamente efficiente e personalizzabile. Gli utenti possono rapidamente passare tra la scrittura di codice, la gestione delle variabili, l'esplorazione dei file e la revisione della cronologia dei comandi, ottimizzando così il flusso di lavoro e aumentando la produttività.

Per quanto riguarda le caratteristiche principali ecco un rapido elenco di tutte le capacità e funzionalità che ci possono interessare maggiormente presenti all’interno di MATLAB:

1. **Toolbox Professionali:** Dispone di toolbox professionalmente sviluppati, accuratamente testati e completamente documentati, che forniscono funzionalità aggiuntive in vari ambiti tecnici e scientifici. Nel nostro caso andremo ad utilizzare i toolbox statistics e machine learning e deep learning toolbox (approfondiremo nel prossimo sottocapitolo)
2. **Interfaccia Intuitiva:** La sua interfaccia grafica utente (GUI) è intuitiva e facile da usare, rendendo l'apprendimento e l'uso del software più accessibili, specialmente per i principianti.
3. **Conversione Automatica di Algoritmi in Codice:** MATLAB è in grado di convertire automaticamente i propri algoritmi in codice C/C++ e HDL, rendendoli pronti per l'implementazione su dispositivi embedded.
4. **Integrazione con Simulink:** Lavora in sinergia con Simulink per supportare la progettazione basata su modelli, utile per la simulazione in vari domini, la generazione automatica di codice e il testing e la verifica di sistemi embedded.

Un aspetto davvero importate di questo software è che gode di una vasta comunità di utenti e sviluppatori, offrendo un ampio range di tutorial, documentazione, forum e risorse online per supporto e apprendimento. Questo dà la possibilità anche per chi si trova alle prime armi di poter comprendere e applicare fin da subito tutti gli strumenti presenti al suo interno a seconda dello scopo di interesse.

5.2 Toolbox

MATLAB, come abbiamo visto, è uno degli ambienti di programmazione e analisi numerica più popolari, ed è sicuramente noto per la sua versatilità e capacità di gestire compiti complessi in vari campi dell'ingegneria, della scienza e della matematica. Un elemento chiave di questa versatilità sono i "toolbox" presenti in MATLAB, che sono essenzialmente pacchetti di funzioni, librerie e strumenti specifici per determinate applicazioni o discipline.

Un toolbox in MATLAB è una raccolta di funzioni, interfacce utente e processi automatizzati progettati per specifici compiti o applicazioni. Questi toolbox estendono le capacità di MATLAB in settori specifici come l'elaborazione dei segnali, l'analisi statistica, l'ottimizzazione, il calcolo simbolico, il controllo dei sistemi, e molti altri. Ogni toolbox è sviluppato per fornire strumenti e funzioni che non sono inclusi nel pacchetto MATLAB standard.

Uno degli obiettivi principali dei toolbox di MATLAB è l'efficienza. Attraverso l'automazione di compiti ripetitivi e complessi, questi strumenti riducono significativamente il tempo e lo sforzo richiesti per l'analisi dei dati e la modellazione. Questo aspetto è fondamentale in ambienti dove il tempo è un fattore critico, come in progetti di ricerca e sviluppo rapidi.

Per facilitarne l'utilizzo, essi possono includere:

1. **Funzioni Specializzate:** Le funzioni specializzate nei toolbox di MATLAB sono progettate per eseguire compiti specifici in un determinato campo. Queste funzioni sono il cuore di ogni toolbox nonché estremamente versatili. Molti toolbox includono algoritmi all'avanguardia che sarebbero difficili da implementare da zero. Inoltre, gli utenti possono spesso personalizzare queste funzioni per adattarle alle loro esigenze specifiche, come modificare i parametri in un modello di machine learning nel Statistics and Machine Learning Toolbox.

2. **Interfacce Utente Grafiche (GUI):** Le GUI nei toolbox di MATLAB migliorano l'usabilità delle funzioni complesse, consentendo agli utenti di interagire con le funzioni senza scrivere codice, rendendo il toolbox

accessibile anche a chi non è esperto di programmazione. Molte GUI includono strumenti di visualizzazione che aiutano a comprendere meglio i dati e i risultati delle analisi. Ad esempio, il Deep Learning Toolbox offre GUI per visualizzare le reti neurali e monitorare il loro training (successivamente lo vedremo nel nostro caso applicato). In fine, di estrema utilità è che gli utenti possono configurare parametri e impostazioni attraverso interfacce intuitive.

3. **Esempi e Dati di Esempio:** Questa componente è fondamentale per l'apprendimento e la comprensione a fondo del toolbox di nostro interesse. Essi spesso includono script di esempio che dimostrano come utilizzare le funzioni in scenari reali. Questi esempi possono servire come punto di partenza per progetti personalizzati. Molti toolbox forniscono set di dati di esempio che permettono agli utenti di testare e esplorare le funzioni senza dover cercare o creare dati da soli. Per esempio, il Deep Learning Toolbox mette a disposizione degli utenti le più recenti tecniche di intelligenza artificiale, consentendo loro di implementare reti neurali avanzate con facilità.

4. **Documentazione Integrata:** La documentazione integrata è un aspetto cruciale che aumenta notevolmente il valore dei toolbox. La documentazione spesso include guide dettagliate su come utilizzare le funzioni del toolbox, con spiegazioni teoriche e pratiche. Oltre alle istruzioni operative, la documentazione può includere tutorial, webinar, e FAQ per aiutare gli utenti a familiarizzare con il toolbox.

Ogni toolbox è unico del suo genere. La prima e più evidente differenziazione che esiste tra i vari toolbox di MATLAB si trova nel loro campo di applicazione. Ogni toolbox è meticolosamente sviluppato per una specifica disciplina o settore.

Questa specializzazione garantisce che gli utenti abbiano accesso a strumenti altamente raffinati e specifici per il loro campo di studio o lavoro. Ogni toolbox si

distingue anche per le funzionalità che offre. Queste funzioni sono sviluppate per rispondere alle esigenze specifiche.

In conclusione, i toolbox di MATLAB rappresentano un elemento fondamentale dell'ecosistema MATLAB, offrendo soluzioni efficienti, specializzate, innovative e facilmente accessibili. Questi strumenti ampliano notevolmente le possibilità di analisi e modellazione, portando a nuovi livelli di produttività e creatività nel campo dell'ingegneria e della scienza.

5.2.1 Deep learning toolbox

Entriamo ora nel dettaglio dei toolbox di nostro interesse, andando ad analizzare come si compongono e cosa possono offrirci. MATLAB è dotato del Deep Learning Toolbox, che offre algoritmi, funzioni e applicazioni per sviluppare, allenare, visualizzare e simulare reti neurali. Questo strumento consente di effettuare operazioni come classificazione, regressione, clustering, riduzione dimensionale, previsione di serie temporali e modellazione di sistemi dinamici. Approfondendo le sue caratteristiche principali, possiamo avere una visione più dettagliata delle sue potenzialità:

- **Progettazione di Reti Neurali:** Il toolbox permette agli utenti di costruire varie architetture di rete neurali. Fra quelle citate nei capitoli precedenti ritroviamo:
  - Reti Neurali Convoluzionali (CNN): Specializzate nell'elaborazione di immagini e video, le CNN sono ottimizzate per riconoscere pattern visivi direttamente dai pixel delle immagini, con applicazioni che vanno dalla diagnostica medica all'analisi di video.
  - Reti Neurali Ricorrenti (RNN): Ideali per lavorare con dati sequenziali e temporali, le RNN sono impiegate in compiti come la previsione di serie temporali, il riconoscimento vocale e la generazione di testo.
  - Multi layer perceptron (MLP): Come già detto in precedenza, la nostra rete si baserà su questa architettura e successivamente andremo ad analizzare tutti i passaggi effettuati per la sua creazione, inizializzazione, training e test.

Queste architetture possono essere ulteriormente personalizzate con strati, funzioni di attivazione e metodi di ottimizzazione specifici per soddisfare requisiti unici del progetto.

- **Integrazione con apprendimento automatico:** Il toolbox integra reti neurali con metodi tradizionali di machine learning, consentendo agli utenti di combinare la potenza del deep learning con tecniche statistiche consolidate, ottimizzando l'analisi di complessi set di dati.
- **Training e Valutazione di Modelli:** Include supporto per diversi algoritmi di ottimizzazione, come la discesa stocastica del gradiente e l'Adam optimizer. Strumenti per validare la robustezza del modello e per sintonizzare gli iperparametri, garantendo l'efficacia e l'affidabilità del modello nonché metodi per valutare accuratamente i modelli tramite metriche come l'accuratezza, la precisione e il richiamo.
- **Sfruttamento di Reti Pre-addestrate:** Fornisce accesso a reti neurali pre-addestrate (come AlexNet, VGG, ResNet), che possono essere adattate a nuovi compiti, riducendo drasticamente i tempi e le risorse necessarie per lo sviluppo.
- **Visualizzazione Intuitiva:** Strumenti per visualizzare l'architettura della rete, il processo di training e i risultati in modi che rendono più semplice interpretare e perfezionare i modelli di deep learning.

In sintesi, il Deep Learning Toolbox di MATLAB rappresenta un ambiente estremamente potente e flessibile per lo sviluppo di soluzioni di deep learning. Con le sue funzionalità avanzate, offre agli utenti gli strumenti per sfruttare appieno il potenziale del deep learning, dall'elaborazione di dati complessi alla costruzione di modelli innovativi.

5.2.2 Statistics and machine learning

Questo toolbox è un pilastro per l'analisi dei dati e la modellazione statistica. Fornisce una vasta gamma di strumenti per l'analisi esplorativa dei dati, la modellazione statistica, l'apprendimento automatico e la simulazione. Con funzionalità che vanno dalla regressione lineare e logistica alla classificazione e al clustering, il toolbox è indispensabile per professionisti e ricercatori che lavorano con grandi set di dati e hanno bisogno di strumenti affidabili per interpretare le loro informazioni, fare previsioni o costruire modelli robusti. Vediamo nel dettaglio le sue principali caratteristiche:

- Metodi Statistici Avanzati: Offre strumenti per analisi esplorative come il calcolo di media, mediana e deviazione standard. Strumenti per costruire modelli che possono prevedere risultati futuri basandosi sui dati, utilizzando tecniche come la regressione lineare e non lineare.
- Algoritmi di Machine Learning: Il toolbox include algoritmi come macchine a vettori di supporto (SVM), foreste casuali e alberi decisionali per compiti di classificazione e regressione. Fornisce inoltre algoritmi come k-means e clustering gerarchico per identificare gruppi o cluster in set di dati.
- Validazione e Selezione del Modello: Già visto in precedenza.
- Gestione di Big Data: Capacità di gestire e analizzare dataset di grandi dimensioni, essenziale nell'era del big data. In aggiunta utilizza il calcolo parallelo e distribuito per accelerare l'analisi di dati voluminosi.
- Visualizzazione dei Dati: Anche questo punto visto in precedenza

In conclusione, il "Statistics and Machine Learning Toolbox" di MATLAB è un insieme estremamente versatile e potente di strumenti per l'analisi statistica e l'apprendimento automatico. Offre agli utenti la capacità di affrontare sfide analitiche complesse in una vasta gamma di settori, rendendo l'analisi dei dati più accessibile, efficiente e significativa.

5.3 Sviluppo del progetto

Come accennato nell'introduzione di questa tesi, l'obiettivo di questa trattazione è quella dell'ottenimento degli angoli di uscita di un quadrilatero articolato (conoscendone la struttura) al variare dell'angolo di ingresso mediante una rete neurale multistrato correttamente addestrata. Il flusso di progetto di progettazione della rete neurale prevede sette fasi principali. Analizzeremo ogni step andando a commentare cosa prevede quel determinato passaggio e come è stato svolto in questa trattazione specifica:

1. Raccolta dati
2. Creare e Configurare la rete
3. Inizializzare i pesi e le polarizzazioni
4. Addestrare e convalidare la rete
5. Utilizzare la rete

La raccolta dei dati nella fase 1 avviene generalmente (come nel nostro caso) al di fuori del software Deep learning Toolbox.
Ogni fase verrà analizzata a parte la fase 3 dato che non necessitavamo di inizializzare pesi particolare, si è scelto di lasciare le impostazioni predefinite.

5.3.1 Raccolta dati

Per quanto riguarda il database utilizzato per l'addestramento della nostra rete neurale abbiamo scelto di utilizzare lo stesso MATLAB per ottenerlo poiché, come vedremo, è stato necessaria la stesura di un codice per l'ottenimento degli angoli di ingresso ma in particolare di quelli di uscita poiché questi ultimi sono funzione di molte variabili inserite in equazioni non lineari. Come si può evincere dal codice in figura 15, oltre alla scrittura del codice per l'ottenimento degli angoli, abbiamo aggiunto la funzione "plot" per visionare graficamente tutte le configurazioni del nostro quadrilatero, da 0 a 360, con lo step di ogni angolo scelto in base alla mole di dati che si vuole utilizzare per l'addestramento della rete.

```
l1 = 4;
l2 = 9;
l3 = 6;
lt = 10;

Xa = 0;
Ya = 0;
Yd = 0;
Xd = lt;

figure; % Aprire una nuova figura
hold on; % Mantenere tutti i plot

th = [];

for TH1 = 0:0.5:360

    A = 2*l2*(l1*cosd(TH1)-lt);
    B = 2*l1*l2*sind(TH1);
    C = (l1*cosd(TH1)-lt)^2 + (l1*sind(TH1))^2 +l2^2 - l3^2;

    TH2 = 2*atand((-B+sqrt(A^2 + B^2 - C^2))/(C-A));
    TH3 = atan2d(-(l1*sind(TH1)+l2*sind(TH2)), -(l1*cosd(TH1)-lt+l2*cosd(TH2)));

    Xb = l1*cosd(TH1);
    Yb = l1*sind(TH1);
    Xc = l1*cosd(TH1) + l2*cosd(TH2);
    Yc = l1*sind(TH1) + l2*sind(TH2);

    th = [th; [TH1, TH2, TH3]];
    plot([Xa,Xb,Xc,Xd], [Ya,Yb,Yc,Yd], 'm')



end

hold off; % Rilascia il grafico per evitare di sovrapporre plot futuri
axis equal; % Imposta uguale scala degli assi per mantenere le proporzioni
grid on; % Aggiunge una griglia per una migliore leggibilità
```

Figura 15 Scrittura codice per angoli di ingresso/uscita quadrilatero e plot dei dati

Per quanto riguarda il codice scritto nel ciclo “For” per l’ottenimento degli angoli possiamo fare alcune improntanti considerazioni. TH1 corrisponde al nostro angolo di ingresso e, come detto in precedenza, la scelta è stata quella di andare a prendere tutti gli angoli (mediante il comando “:”) compresi fra 0 e 360 gradi con step di 0.5, andando ad ottenere un primo set di dati composto da 721 elementi. Risolvendo le equazioni di chiusura del meccanismo è possibile ottenere le espressioni analitiche degli altri due angoli TH2, TH3. Per risolvere queste equazioni è necessario l’utilizzo delle variabili di appoggio A, B, C. Nel nostro caso, l’equazione di TH2 è la seguente poiché è stato verificato che A è sempre diverso da C per ogni configurazione, dunque non sono presenti criticità. L’utilizzo di “cosd”, “sind” e “tand” è legata al fatto che, se non

avessimo usato queste funzioni in questa scrittura, avremmo dovuto inserire, e in seguito avremmo ricevuto, valori in radianti. Ciò per la nostra applicazione non è congeniale poiché è estremamente poco intuitivo. Così facendo otteniamo e riceviamo gli angoli espressi in gradi, rendendo più semplice e rapida la lettura dei valori

Il plot delle configurazioni del quadrilatero articolato è stato ottenuto mediante il comando "plot" (comando base di MATLAB) delle posizioni Xa, Xb, Xc, Xd e Ya, Yb, Yc, Yd ossia le posizioni sull'asse cartesiano X, Y dei punti A, B, C, D che coincidono con le coppie rotoidali (ossia dove si uniscono le coppie di aste del quadrilatero). Le lunghezze delle 4 aste sono state scelte nel rispetto della condizione di Grashof: M + m ≤ $i1$ + $i2$ (con M e m asta più lunga e più corta mentre $i1$ e $i2$ di lunghezza intermedia), in modo tale da avere almeno una manovella. Le lunghezze che e i punti che rimangono fissi sono stati riportati in alto poiché costanti. Per quanto riguarda invece le espressioni degli altri punti esse sono state ottenute mediante equazioni scritte a partire dal riferimento coincidente con il punto A.

Per ottenere la sovrapposizione di tutti i plot i un'unica immagine (figura 16) è stato utilizzato il comando "hold on" e per ottenere le griglie per dare più comprensione dell'immagine è stato usato il comando "grid on".

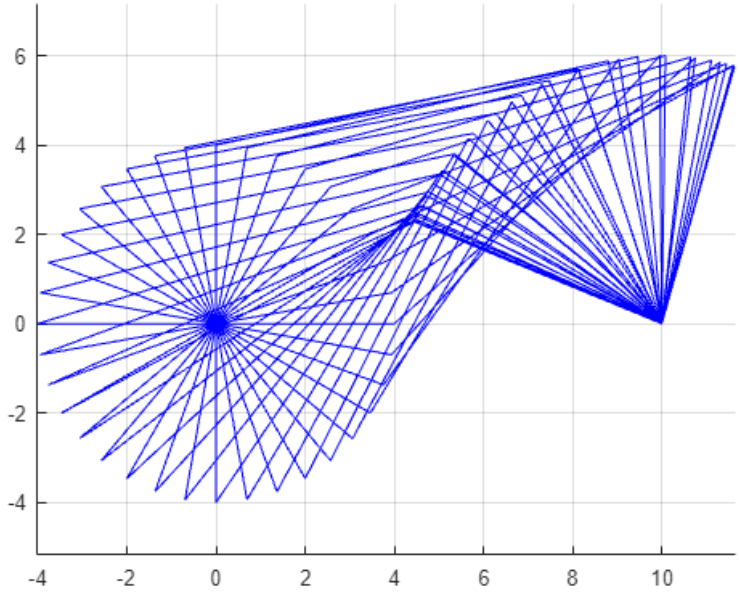


Figura 16 Plot di tutte le configurazioni del quadrilatero con meno valori di quelli usate per l'addestramento per mostrare l'andamento

5.2.2 Creazione e configurazione della rete

Veniamo ora alla fase forse più importante ed interessante della trattazione, l'effettiva creazione della rete neurale.

Come detto in precedenza, la rete scelta per questo elaborato è una rete feed for ward, in particolare, una multy-layer perceptron. La rete feedforward multistrato è il cavallo di battaglia del Deep learning.

Come si può notare, alla riga 8 del codice in figura 17 abbiamo usato la funzione "feedforwardnet" per creare una rete feedforward multistarto. Se questa funzione viene chiamata senza argomenti in entrata, si crea in automatico una rete predefinita non configurato. La rete può essere configurata con il comando "configure".

```
1  % Passaggio 1: Preparazione dei Dati
2  % Supponiamo di avere un set di dati di input (angoli di ingresso) e output (angoli di uscita)
3  input_angles = THin; % Inserisci qui i dati degli angoli di ingresso
4  output_angles = THout; % Inserisci qui i dati degli angoli di uscita
5
6  % Creiamo una rete feedforward con 1 strato nascosto di 20  e algortimo di
7  % retropropagazione di Levengerg-Marquardt
8  net = feedforwardnet(20, 'trainlm');
9
10 % Passaggio 3: Configurazione dell'Addestramento
11 net.divideParam.trainRatio = 70/100; % 70% dei dati per l'addestramento
12 net.divideParam.valRatio = 15/100; % 15% dei dati per la validazione
13 net.divideParam.testRatio = 15/100; % 15% dei dati per il test
14
15 % Scegliamo la funzione di attivazione tan-sigmoid per lo strato nascosto
16 net.layers{1}.transferFcn = 'tansig';
17
18 % Addestramento della Rete
19 [net, tr] = train(net, input_angles, output_angles);
```

Figura 17 Creazione rete neurale, configurazione e training

Il primo elemento fra parentesi è una matrice contenente il numero di neuroni in ogni strato nascosto. Predefinito, il numero di neuroni è 10, ma è possibile modificare questo valore a seconda delle necessità di calcolo. Una nota importante, emersa per via empirica, è che un numero di neuroni elevato non sempre è la soluzione più adatta poiché c'è il rischio di mettere in difficoltà in calcolatore portando all'overfitting (già trattato nei capitoli precedenti) ottenendo risultati corrotti. Nel nostro caso si è optato per 20 poiché è un giusto compromesso fra potenza di calcolo e tempistiche di

esecuzione. Per lo stato nascosto si è optato per la funzione di attivazione “tansig” (linea 16 del codice in figura 17) mentre per lo strato di output si è lasciata la funzione di attivazione predefinita per questo stato, ossia la funzione di trasferimento lineare pureline che è considerata la migliore.

E’ possibile aumentare il numero di starti nascosti da 1 a 2 ma tendenzialmente con 1 uno strato si ottengono già risultati eccellenti (1 strato è il valore predefinito). Il secondo argomento indica invece indica la funzione di addestramento da utilizzare per addestrare la rete. Nel nostro caso si è optato per la funzione trainlm. Il nome sta per "train Levenberg-Marquardt" ed è basata sull'algoritmo di ottimizzazione Levenberg-Marquardt. Questo algoritmo è particolarmente noto e ampiamente utilizzato per l'addestramento delle reti neurali feedforward, in particolare per problemi complessi e reti di grandi dimensioni.

Le caratteristiche prinipali di questa funzione sono:

1. **Velocità**: È uno degli algoritmi più rapidi per l'addestramento di reti neurali, specialmente per reti di medie e piccole dimensioni.
2. **Convergenza**: Ha una buona capacità di convergere rapidamente verso una soluzione ottimale, soprattutto in situazioni dove il gradiente della funzione di errore è particolarmente significativo.
3. **Utilizzo della Memoria**: Tende ad essere più esigente in termini di memoria rispetto ad altri metodi, come il gradiente discendente o il gradiente coniugato, a causa del calcolo dell'Hessiano e della sua matrice inversa (o di una sua approssimazione).
4. **Applicazioni**: È particolarmente efficace in applicazioni di apprendimento supervisionato, come la regressione e la classificazione.

Per quanto riguarda invece il suo funzionamento l'algoritmo aggiorna i pesi della rete calcolando l'Hessiano (o un'approssimazione di esso) e il gradiente dell'errore rispetto ai pesi. Inoltre, cerca di minimizzare una funzione di errore, come l'errore quadratico medio, modificando i pesi in modo iterativo.

La particolarità dell'algoritmo Levenberg-Marquardt è che aggiunge un termine di regolarizzazione all'Hessiano per evitare problemi numerici e per controllare la direzione di aggiornamento dei pesi, rendendolo un metodo di regolarizzazione.

Per quanto riguarda la configurazione della rete si è scelto di suddividere in 3 set i dati di ingresso con la suddivisione 70/15/15 ripartiti rispettivamente per addestramento, validazione, test (vedere linea 10 figura 17). Questa scelta è stata effettuata per evitare proprio l'over fitting come descritto nel capitolo dedicato.

5.2.3 Addestramento e convalidazione della rete

Gli angoli per l'addestramento (THin e THout) sono stati estratti dalla matrice degli angoli "th" (fig.18) ottenuta dal codice in figura 15.

3601x3 double

	1	2	3
1	0	41.4096	-97.1808
2	0.1000	41.3430	-97.2473
3	0.2000	41.2763	-97.3136
4	0.3000	41.2097	-97.3798
5	0.4000	41.1432	-97.4457
6	0.5000	41.0766	-97.5113
7	0.6000	41.0101	-97.5768
8	0.7000	40.9436	-97.6420
9	0.8000	40.8771	-97.7070
10	0.9000	40.8107	-97.7718
11	1	40.7443	-97.8363
12	1.1000	40.6779	-97.9006
13	1.2000	40.6116	-97.9647
14	1.3000	40.5453	-98.0286
15	1.4000	40.4790	-98.0922
16	1.5000	40.4128	-98.1556

Figura 18 Matrice 3601x3 degli angoli TH1,TH2 e TH3

Questo passaggio è stato effettuato manualmente nella command window poiché per l'addestramento ci servivano solo i valori di ingresso e uscita. In oltre, era necessario effettuare un trasposta delle matrici (comando " ' ") per ottenere una matrice 1x3601 sia per l'ingresso che per l'uscita(fig.19).

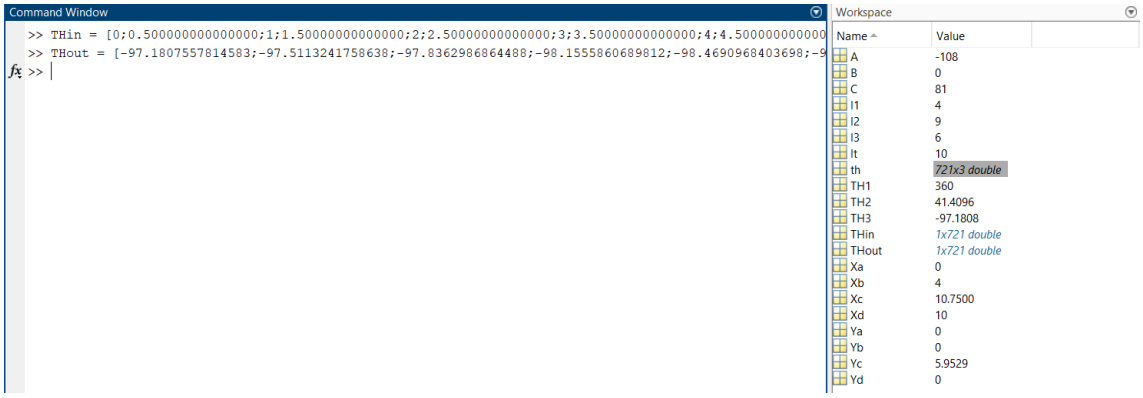


Figura 19 Angoli di ingresso e uscita per l'addestramento

Come anticipato in precedenza, la scelta della funzione di addestramento è ricaduta sull'algoritmo di retropropagazione "trainlm" per i motivi indicati precedentemente.

Il codice scritto per l'addestramento è il seguente:

`[net, tr] = train (net, input_angles, output_angles)` (riga 19, figura 17)

Nell'addestramento delle reti neurali in MATLAB, termini come "epoche", "fattore mu" e "gradiente" sono concetti chiave che svolgono ruoli importanti. Ecco una spiega di ciascuno di essi:

- Epoche: Un'epoca è un singolo passaggio attraverso l'intero set di addestramento. Durante un'epoca, la rete neurale viene addestrata con tutti i dati di addestramento una volta. Più epoche significano che la rete ha più opportunità di apprendere dai dati. Tuttavia, troppe epoche possono portare a un sovradattamento, dove la rete impara a memorizzare i dati di addestramento piuttosto che a generalizzare da essi.
- Fattore Mu: Il "fattore mu" è specifico dell'algoritmo di addestramento Levenberg-Marquardt. È un parametro che aiuta a controllare il mix tra il gradiente discendente e il metodo di Gauss-Newton nell'algoritmo. Quando il fattore mu è grande, l'algoritmo si comporta più come un gradiente discendente puro, utile per superare i minimi locali. Quando è piccolo, si comporta più come il metodo di Gauss-Newton, efficace per convergere rapidamente verso un minimo. MATLAB regola automaticamente il fattore mu durante l'addestramento, aumentandolo o diminuendolo in base al successo degli aggiornamenti dei pesi.

- Gradiente: Il gradiente in una rete neurale si riferisce al gradiente della funzione di errore rispetto ai pesi e ai bias della rete. È un vettore che indica la direzione nella quale la funzione di errore cresce più rapidamente. Nell'addestramento delle reti neurali, si cerca di modificare i pesi e i bias in modo da muoversi nella direzione opposta al gradiente, per ridurre l'errore. Metodi di addestramento come il gradiente discendente utilizzano il gradiente per aggiornare i pesi ad ogni iterazione.

L'addestramento avviene secondo questi parametri che hanno valori predefiniti che riporteremo qua sotto. Per quel che riguarda il nostro addestramento si è optato per lasciare i valori standard:

• net.trainParam.epochs	1000	Numero massimo di epoche
• net.trainParam.goal	0	Obbiettivi di performance
• net.trainParam.max_fail	6	Massimi fallimenti di validazione
• net.trainParam.min_grad	$1e^{-7}$	Minima prestazione del gradiente
• net.trainParam.mu	0.001	mu iniziale
• net.trainParam.mu_dec	0.1	Fattore decrescente di mu
• net.trainParam.mu_inc	10	Fattore incremento mu
• net.trainParam.mu_max	$1e^{10}$	mu massimo
• net.trainParam.show	25	Epoche tra le visualizzazioni

I vettori di convalida vengono utilizzati per fermare l'addestramento in anticipo se le prestazioni della rete sui vettori di convalida non migliorano o rimangono invariati un massimo di epoche fallite consecutivamente.
Giunti a questo punto non resta che mandare in esecuzione il codice mediante il comando "run" e attendere il compimento dell'addestramento del codice.

Ciò che appare subito dopo l'invio dell'addestramento è una immagine (figura 20) dove sono riportati tutti i valori della performance di addestramento con ulteriori plot di diagrammi e tabelle fornite da MATLAB che andremo a visionare man mano.

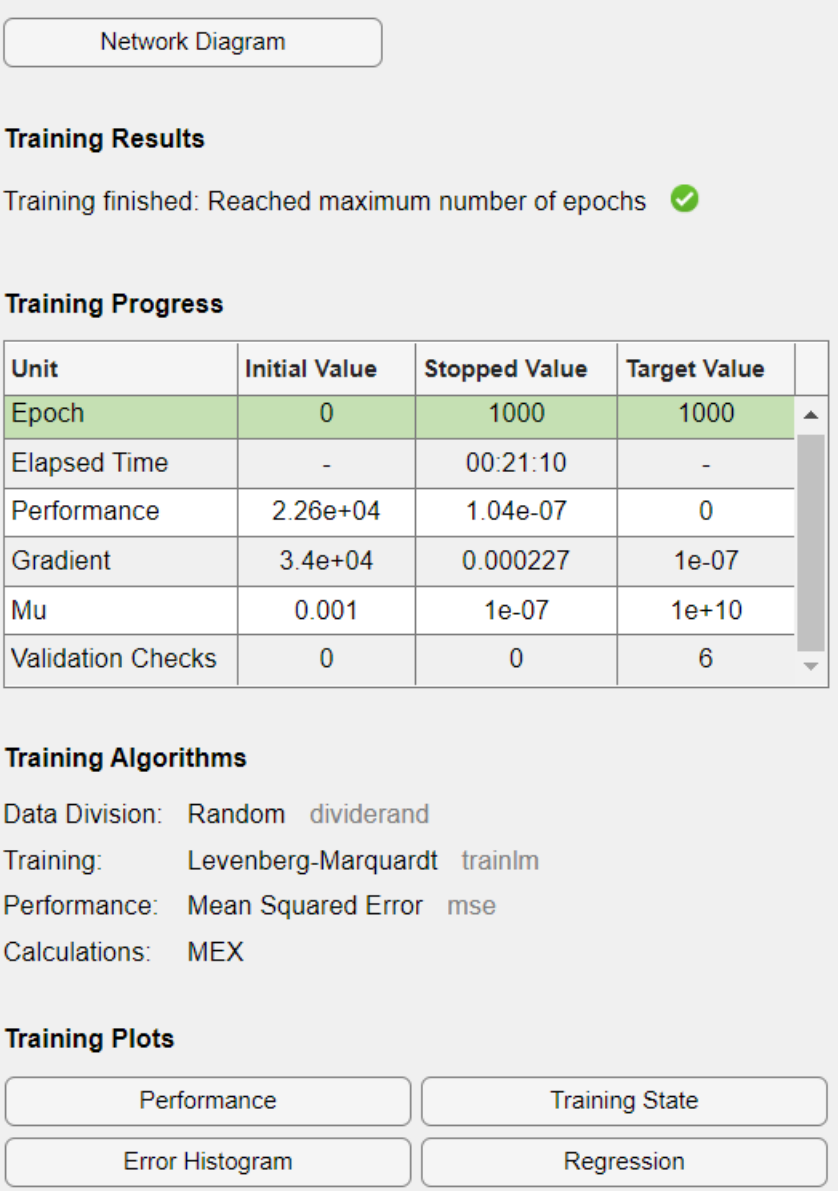


Figura 20 Tabella di addestramento della rete neurale

Come si può notare dalla riga evidenziata in verde si evince che il nostro addestramento è stato interrotto per il raggiungimento massimo del numero di epoche (valore predefinito 1000) e il tempo impegato per l'addestramento è stato di 21 minuti e 10 secondi.

Un tempo cosi elevato è dovuto alla esponente mole di dati introdotti nel training, ben 36001. All'interno del capitolo 6, in cui parleremo delle conclusioni, approfondiremo questo aspetto sulla scelta del numero di valori e come varia la performance in funzione del cambaimento di valori inseriti e numero di neuroni usati nello strato intermedio.

Passiamo ora ad analizzare i vari “training plot” che mette a disposizione il software. Questi sono grafici che MATLAB può generare per visualizzare varie metriche e aspetti dell'addestramento. Includono:

- **Performance**: Mostra come la prestazione cambia durante l'addestramento (figura 21). L’ideale è che ci sia una sovrapposizione dei 3 set di dati e che essi convergano a 0 senza alti e bassi come avvenuto nel nostro caso.

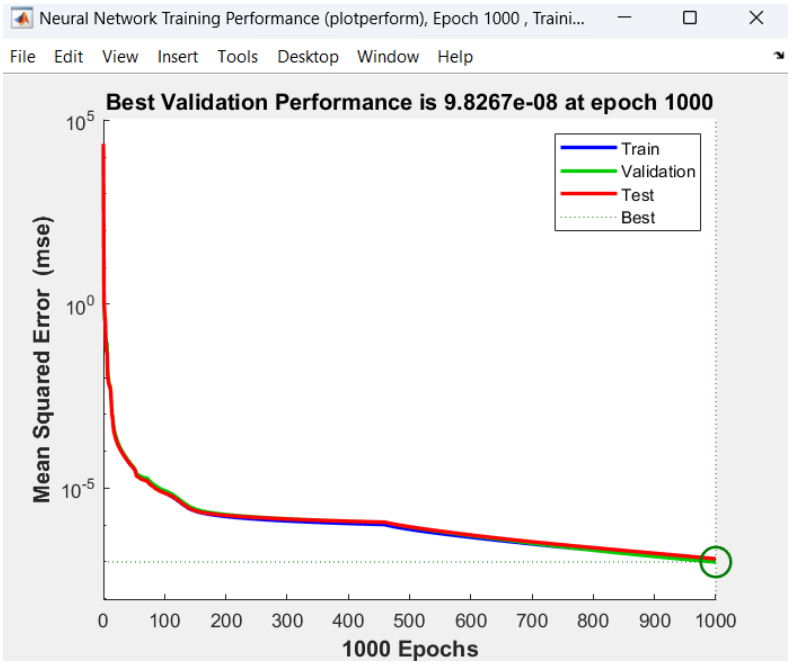


Figura 21 Performance della nostra rete

- **Training State**: Mostra lo stato dell'addestramento, inclusi gradiente e mu (figura 22).

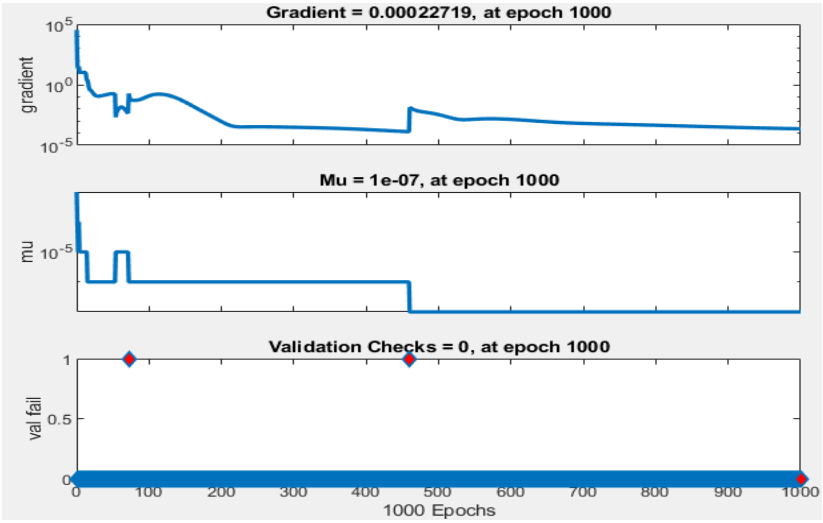


Figura 22 Training state

- **Error Histogram**: Mostra la distribuzione degli errori (figura 23). Anche in questo caso si piò notare un valore medio di errore davvero minimo attorno al 0.0001 di tolleranza.

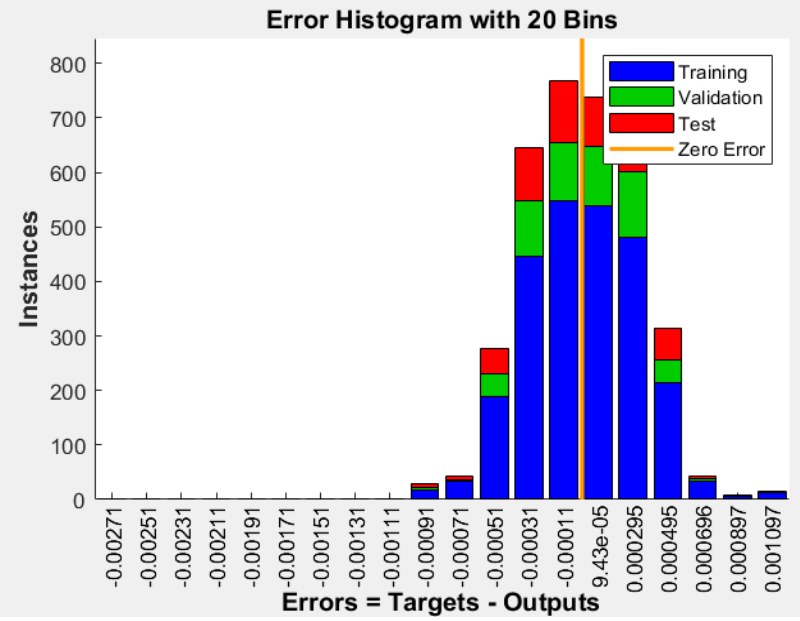


Figura 23 Diapositiva della distribuzione dell valore di errore

- **Regression**: Mostra una comparazione tra gli output della rete e i target desiderati, ideale per valutare la qualità della regressione (figura 24). Livelli ideali si ottengono quando R=1 ad indicare una perfetta correlazione fra i dati forniti.

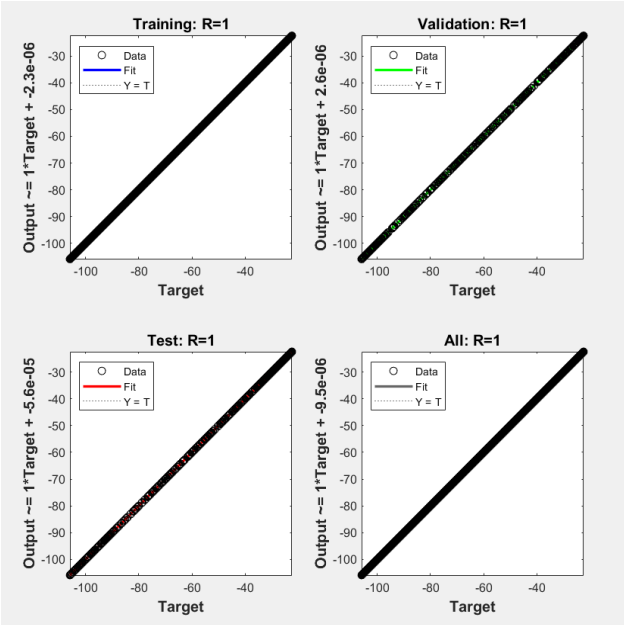


Figura 24 Immagine che riporta la nostra regressione

In conclusione, questo screenshot, mostra che la rete neurale ha completato l'addestramento con successo, raggiungendo un valore molto basso di errore e un piccolo valore di gradiente, indicando che la rete è potenzialmente ben addestrata per il compito assegnato.

5.2.4 Utilizzo della rete

Una volta ottenuta la nostra rete neurale non resta che provarla mediante la command window attraverso il comando “net” ad indicare l’utilizzo della nostra rete neurale. Dopo aver inviato questo comando, il software MATLAB, utilizzando la rete che noi gli abbiamo fornito, restituirà il valore in uscita corrispondente al valore fornitogli in ingresso come nell’esempio mostrato in figura 25.

```
>> p = net(39.1)

p =

 -104.2865
```

Figura 25 Utilizzo della rete addestrata

Mediante il calcolo analitico effettuato a parte emerge che, come riportato da dai grafici visto precentemente, l’errore è apporssimabile al 0.0001. Questo significa che la nostra rete è stata ben addestrata e funzionante.

Capitolo 6

Conclusioni e sviluppi futuri

In conclusione, lo studio presentato in questa tesi ha dimostrato che l'implementazione di reti neurali tramite MATLAB può essere un approccio efficace per risolvere problemi complessi di previsione, come la determinazione degli angoli di uscita di un quadrilatero articolato. Attraverso l'uso di algoritmi di addestramento avanzati, in particolare l'algoritmo di Levenberg-Marquardt, siamo stati in grado di addestrare una rete neurale che mostra una precisione notevole e una buona generalizzazione nei dati di test. La scelta di un data set cosi abbondante è stata effettuata per permettere una perfetta elaborazione di dati senza entrare in overfitting permettendo alla rete di ritracciare tutte le correlazioni fra i dati forniti. E' emerso da ulteriori prove che lo stesso risultato (o molto vicino ad esso) era possibile ottenerlo anche con un numero inferiore di dati, attorno alle 721 unità (step di 0.5 gradi) con un tempo di addestramento decisamente inferiore (6 secondi circa). Per quanto riguarda invece la scelta del numero di neuroni per lo strato intermedio è emerso che fra 10 (numero standard) e 20 vi era una sottile differenza a favore della seconda opzione ,dunque si è deciso di optare per quest'ultima. Tornando a alla discussione del nostro elaborato, è emerso che l'analisi dei risultati ottenuti dimostra che la rete è capace di catturare le relazioni non lineari intrinseche al comportamento del sistema articolato, fornendo previsioni accurate degli angoli di uscita in risposta a vari input. Questo successo è in gran parte dovuto alla scelta accurata della topologia della rete, alla pre-elaborazione dei dati e alla metodologia di addestramento, che insieme hanno creato un modello robusto contro il sovradattamento e con una velocità di convergenza elevata.

Inoltre, l'interfaccia intuitiva e le funzionalità avanzate di MATLAB hanno facilitato la fase di sperimentazione, permettendo di esplorare diverse architetture di rete e parametri di addestramento in modo efficiente. La capacità di visualizzare i risultati attraverso grafici e di analizzare le metriche di prestazione ha contribuito in modo significativo all'ottimizzazione del modello.

Infine, i risultati promettenti ottenuti in questo progetto aprono la strada a ulteriori ricerche. Si potrebbe esplorare l'applicazione di reti neurali più profonde o di algoritmi di apprendimento alternativi per confrontare le prestazioni e, potenzialmente, migliorare ulteriormente l'accuratezza del modello. Inoltre, si potrebbero considerare approcci di addestramento in tempo reale che permetterebbero di adattare dinamicamente i parametri del modello in risposta a nuovi dati o a cambiamenti nelle condizioni operative.

In conclusione, questo lavoro conferma l'efficacia delle reti neurali e di MATLAB come strumento potente per l'analisi e la modellazione di sistemi complessi, offrendo un contributo prezioso sia al campo dell'intelligenza artificiale sia alla meccatronica.

Bibliografia

1. M. Callegari, F. Pellicano, P. Fanghella, Meccanica applicata alle macchine, CittàStudi seconda edizione
2. Documentazione Matlab
3. Chris M. Bishop, Neural networks and their applications, Review of Scientific Instruments / Volume 65 / Issue 6, (1994).
4. Cèsar, Pèrez, Lòpez, Deep learning e reti neurali multistrato. Applicazioni con Matlab
5. Reti neurali [Archivio] - Hardware Upgrade Forum
6. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
7. Bishop, C. M. Pattern recognition and Machine Learning
8. Haykin, S. (1998). Neural Networks: A Comprehensive Foundation*. Prentice Hall.
9. Nielsen, M. A. Neural Networks and Deep Learning. determination Press.
10. MODULO 3: CENNI SULLE RETI NEURALI - University of Cagliari, https://www.unica.it/static/resources/cms/documents/Unit4.pdf.
11. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
12. https://www.guru99.com/backpropogation-neural-network.html#:~:text=Backpropagation%20is%20the%20essence%20of,reliable%20by%20increasing%20its%20generalization
13. https://www.techtarget.com/searchenterpriseai/definition/backpropagation-algorithm#:~:text=Backpropagation%2C%20or%20backward%20propagation%20of,used%20to%20quickly%20calculate
14. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in neural information processing systems, 25.

15. Sci-Hub | Learning representations by back-propagating errors. Nature. Learning representations by back-propagating error. Nature, 323(6088), 533-536.
16. Networkdigital360
17. https://datascience.eu/it/apprendimento-automatico/unita-ricorrenti-gated-gated-recurrent-units/

Ringraziamenti

Nel concludere questo importante capitolo della mia vita accademica, desidero esprimere la mia più profonda gratitudine a tutte le persone che hanno reso possibile questo viaggio.

Innanzitutto, vorrei ringraziare il Professor **Matteo Claudio Palpacelli**, la cui disponibilità e serenità che mi ha trasmesso in tutto l'arco del tirocinio e scrittura della tesi sono state fondamentali nel percorso di realizzazione di questo lavoro.

Un grazie speciale va alla mia famiglia. Ai miei genitori, **Gabriele Savoia** e **Baldazzi Sabrina**, per il sostegno e la fiducia che hanno sempre riposto in me. Il vostro sostegno ha plasmato la persona che sono oggi e senza di voi, questo traguardo non sarebbe stato possibile.

Desidero inoltre esprimere la mia riconoscenza a tutti gli amici e colleghi che mi hanno accompagnato in questo percorso. Il vostro supporto, i momenti condivisi e le discussioni stimolanti sono stati un arricchimento personale e professionale che porterò sempre con me.

Infine, un ringraziamento a tutte quelle persone che, anche se non menzionate direttamente, hanno contribuito in piccole ma significative maniere a questo mio percorso. Ogni incontro, conversazione ed esperienza con voi è stata un prezioso tassello nel mosaico di questa mia avventura.

Con sincera gratitudine,

Savoia Davide