Politecnico di Torino

Politecnico di Torino

Corso di Laurea Magistrale in Ingegneria Meccanica
a.a. 2021/2022
Sessione di Laurea Luglio 2022

Sviluppo di una metodologia iterativa per la risoluzione di problemi di ottimizzazione tramite reti neurali

Relatrice:
Prof.ssa Cristiana Delprete

Correlatore:
Ing. Giovanni Pesare

Candidato:
Edoardo Compagnone
matricola 278990

A chi è felice per questo traguardo,
e a chi lo sarebbe stato...

Sommario

Nell'ambito della progettazione meccanica assume un ruolo fondamentale l'ottimizzazione strutturale: infatti è sempre più richiesto il miglioramento delle prestazioni nel rispetto dei vincoli di progetto.

Al momento esistono numerosi software che consentono di effettuare ottimizzazioni di vario genere: parametrica, topologica e di forma. Gli applicativi commerciali non sempre garantiscono la risoluzione di problemi complessi. Sono un esempio quelli caratterizzati da variabili discrete e funzioni obiettivo discontinue, oppure dall'interazione tra diverse discipline (MDO).

Con l'obiettivo di venire incontro a questa crescente richiesta del mercato, è stata concepita una metodologia che accoppia il processo di ottimizzazione alle tecniche di intelligenza artificiale. Nello specifico, questa si avvale delle reti neurali per predire con precisione gli output dell'analisi in sostituzione del solutore commerciale. Le reti, una volta addestrate, vengono integrate all'interno di un algoritmo di ottimizzazione euristico, ispirato alla selezione naturale e alla teoria dell'evoluzione. L'intero procedimento è stato reso automatico e l'intervento dell'utente è necessario solamente per la definizione di alcuni parametri iniziali.

Nel capitolo introduttivo vengono illustrati i problemi di ottimizzazione, la loro classificazione e i driver fondamentali per la scelta dell'algoritmo. I metodi sono suddivisi principalmente in due grandi gruppi, quelli basati sul calcolo del gradiente (gradient-based) e quelli che invece utilizzano altre tecniche (gradient-free). Successivamente vengono presentate le reti neurali, ovvero modelli di calcolo matematico-informatici che apprendono seguendo meccanismi tipici dell'intelligenza umana. Nello specifico, viene effettuato un focus sulla loro struttura e sulle modalità di addestramento.

Si prosegue con la presentazione del problema oggetto dell'ottimizzazione, impiegato come caso di studio per lo sviluppo del codice. Il modello è stato realizzato in Altair Hypermesh, utilizzando Altair Optistruct come solutore per risolvere l'analisi strutturale. Optistruct fornisce anche la possibilità di risolvere il problema di ottimizzazione proposto, permettendo la validazione dell'intera metodologia sviluppata.

Il corpo centrale dell'elaborato è dedicato alla spiegazione delle sezioni in cui è articolato il codice, suddiviso in 3 step. Il primo è relativo alla creazione del database, contenente gli input e gli output delle analisi. La seconda fase è dedicata alla definizione dei parametri caratteristici della rete neurale e al successivo addestramento. Il terzo step riguarda il settaggio dell'algoritmo di ottimizzazione, stabilendo la funzione da minimizzare e i vincoli di progetto. Al fine di migliorare l'affidabilità delle reti addestrate e, contestualmente, del processo di ottimizzazione implementato, il codice è stato sviluppato in tre versioni di complessità graduale. Tutti i risultati ottenuti durante il processo di raffinamento del codice vengono presentati e confrontati tra loro, basando le valutazioni sui tempi di processo e sulla precisione della soluzione.

La tesi procede con la risoluzione di due problemi di ottimizzazione non risolvibili con software commerciali. Il primo è caratterizzato da variabili di progetto discrete, mentre il secondo abbina un'analisi economica alla verifica strutturale. La metodologia sviluppata ha permesso di far fronte ad entrambe le casistiche.

In conclusione, sono riassunti i principali punti di forza della metodologia proposta, correlandoli con i limiti riscontrati nell'utilizzo dei tool commerciali per la risoluzione di problemi complessi e con i risultati ottenuti. Infine, vengono presentate alcune proposte di sviluppo del codice, che permettano di migliorare l'efficienza del processo.

Abstract

Structural optimization assumes a fundamental role in the field of mechanical design: in fact the performance improvement, subjected to the design constraints, is increasingly in demand.

Currently there are lot of software that allow to perform various kinds of optimization: size, topology and shape. Commercial applications not always guarantee the resolution of complex problems. For example those characterized by discrete variables and discontinuous objective functions, or by the interaction among different disciplines (MDO).

To meet this market demands, a methodology that couples optimization process and artificial intelligence techniques has been devised. In particular, it makes use of neural networks to accurately predict the analysis outputs in place of the commercial solver. Once trained, the networks are integrated within a heuristic optimization algorithm, which is inspired by natural selection and the theory of evolution. The entire process has been automated and the user intervention is only necessary for defining some initial parameters.

Optimization problems, their classification and the fundamental drivers for the algorithm choice are illustrated in the introduction chapter. The methods are mainly divided into two big groups, those based on gradient calculation (gradient-based) and those using other techniques (gradient-free). After, that neural networks are presented, i.e. mathematic-computer models the learn by following mechanisms typical of human intelligence. In detail a focus is made on their structure and training rules.

It continues with the presentation of the optimization problem, which has been used as a case study for the development of the code. The model has been implemented in Altair Hypermesh, using Altair Optistruct to solve the structural analysis. Optistruct also provides the possibility of solving the proposed optimization problem, allowing the validation of the entire developed methodology.

The main body of the paper is dedicated to explaining the 3 steps into which the code is divided. The first relates to the creation of the database, containing the inputs and outputs of the analyses. The second phase is dedicated to the definition of the characteristic parameters of the neural network and the subsequent training. The third step concerns the setting of the optimization algorithm, establishing the function to be minimized and the design constraints. In order to improve the reliability of the trained networks and, at the same time, of the implemented optimization process, the code has been developed in three versions of gradual complexity. All the results obtained during the code refinement process are presented and compared with each other, basing evaluations on the process time and accuracy of the solution.

The thesis proceeds by solving two optimization problems that cannot be solved with commercial software. The first is characterized by discrete design variables, while the second combines an economic analysis with structural verification. The developed methodology has allowed to cope with both cases.

In conclusion, the main strengths of the proposed methodology are summarized, correlating them with the limitations encountered in using commercial tools to solve complex problems and with the results obtained. Finally, some code development proposals are presented, to improve the efficiency of the process.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Introduzione all'ottimizzazione

L' ottimizzazione fa parte dell'istinto umano ed è un concetto che applichiamo nella vita di tutti i giorni, per migliorare determinate azioni o sistemi che ci circondano. Spesso viene associata alla realizzazione di un miglioramento, ma in termini matematici è esplicabile tramite un concetto molto più preciso: *"ricerca della migliore soluzione possibile cambiando le variabili che vengono controllate, spesso sottostando a determinati vincoli"* [14].

Si tratta di una filosofia applicabile in diversi ambiti, che riguardano ad esempio l'ingegneria, l'economia, la fisica, la chimica, la biologia. All'interno di questa tesi verrà approfondito il tema dell'ottimizzazione in ambito ingegneristico, in particolar modo applicata alla progettazione meccanica.

Durante la fase di sviluppo di un prodotto, vengono eseguite determinate operazioni in successione, annidate all'interno di un processo iterativo. Considerando un approccio classico, infatti, una volta definite le specifiche, viene realizzato il progetto preliminare. A questo punto sono previste una serie di iterazioni, caratterizzate dalla valutazione del modello e dalla modifica dello stesso sulla base di studi e conoscenze pregresse. Una volta che la soluzione è considerata soddisfacente, viene realizzato il progetto definitivo. Questo approccio, oltre che ad essere lento e macchinoso, non assicura che la soluzione trovata corrisponda alla migliore possibile, considerando che le decisioni vengono prese grazie all'intuizione dei progettisti. Spesso in questi casi si considera per buona una soluzione che rispetti i vincoli imposti, ignorando la possibilità che vi sia una casistica ancora più valida.

L'ottimizzazione ha proprio lo scopo di rendere questo processo più rapido e di ridurre l'intervento dell'uomo nella valutazione dei vari step, conducendo ad un risultato migliore. L'ingegnere non ha più il compito di valutazione e modifica del progetto durante le varie iterazioni, ma, oltre alla realizzazione del progetto iniziale, deve occuparsi dell'impostazione del problema di ottimizzazione. Nel corso del processo, il controllo e la modifica delle variabili di progetto vengono effettuati in automatico, fino al raggiungimento della soluzione che soddisfa le condizioni ottimali, assicurando che nessun altra configurazione sia più vicina all'ottimo.

Per poter realizzare un processo di questo tipo, è necessario che l'ingegnere sia esperto in ambito tecnico per poter formulare il problema e determinare le specifiche; è richiesta inoltre la conoscenza di algoritmi e calcolo numerico per poter impostare in modo efficacie l'ottimizzatore. Al termine del processo è fondamentale che il progettista valuti la fattibilità della soluzione proposta dall'algoritmo di ottimizzazione. In caso di esito negativo bisognerà quindi apportare modifiche al progetto iniziale o riformulare il problema.

Nell'impostazione di un problema di ottimizzazione è necessario definire:

- **x**: il vettore delle variabili di progetto, chiamata anche soluzione incognita
- **Objective Function**: funzione scalare di x che si vuole minimizzare. Quella della funzione obiettivo è una scelta cruciale perché deve rappresentare al meglio il fine dell'ottimizzazione
- **Constraints**: funzioni scalari di x che determinano uguaglianze o disuguaglianze che il vettore incognito deve soddisfare. Questi vincoli sono fondamentali per far sì che la soluzione sia realistica e accettabile
- **Bounds**: vincolo che agisce direttamente sulle variabili di progetto, limitando il campo di soluzioni

Una volta illustrate queste definizioni, possiamo dire che un problema di ottimizzazione consiste nel: *minimizzare la Objective Function variando le variabili di progetto rispettando i Constraints e i Bounds.*
In termini matematici [14]:

$$\begin{array}{rll} minimizzare & f(x) & \\ variando & \underline{x_i} \leq x_i \leq \overline{x_i} & i = 1, ..., n_x \\ rispettando & g_j(x) \leq 0 & j = 1, ..., n_g \\ & h_k(x) = 0 & k = 1, ..., n_h \end{array}$$

1.1.1 Classificazione dei problemi di ottimizzazione

Per ottenere un'ottimizzazione soddisfacente è necessario scegliere l'algoritmo in base alle caratteristiche del problema. Le principali classificazioni sono [17]:

- Ottimizzazione continua o discreta, in relazione al fatto che alcune variabili hanno senso solo se intere. In ambito di produzione, ad esempio, alcuni componenti sono realizzabili solo in determinate taglie e dimensioni, ad intervalli ben precisi. In questi casi, approssimare la soluzione derivante da un'ottimizzazione effettuata su valori continui potrebbe comportare la violazione dei vincoli o l'allontanamento dall'ottimo;

- Problemi lineari o non-lineari, in relazione alla tipologia delle funzioni che definiscono l'obiettivo e i vincoli;

- Ottimizzazione globale o locale, dipendente dalla complessità della funzione obiettivo, in particolare dalla presenza di più minimi locali;
- Ottimizzazione deterministica o stocastica, in base alla presenza o meno di quantità sconosciute al momento della formulazione del problema.

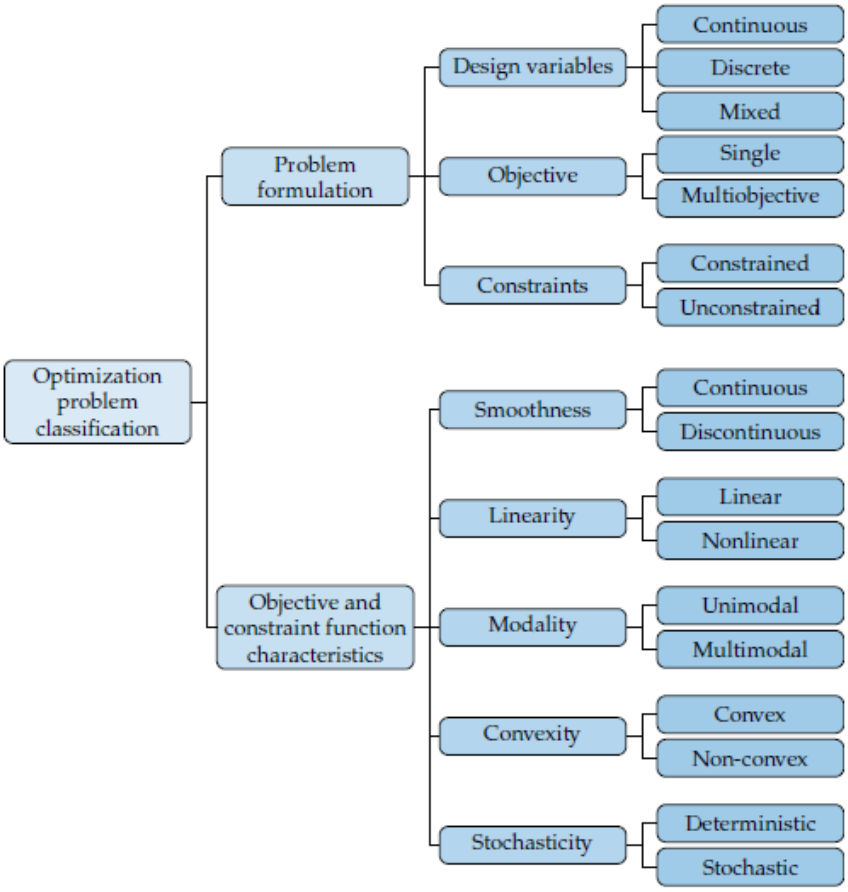

Figura 1.1: Classificazione dei problemi di ottimizzazione (tratta da *Engineering design optimization* [14])

Quando si tratta di problemi relativamente semplici, la ricerca dell'ottimo viene effettuata tramite metodi basati sul calcolo del gradiente. Si tratta del vettore costituito dalle derivate parziali della funzione, calcolato tramite rapporti incrementali lungo le direzioni degli assi delle variabili. Essendo diretto verso la direzione lungo cui cresce la funzione, ne permette di valutare l'andamento all'interno del suo dominio, fino al raggiungimento del minimo. In funzione delle caratteristiche del problema, esistono diversi metodi, tra cui: Steepest Descent, Conjugate Gradient, Nonlinear Conjugate Gradient, Newton, Modified Newton, Quasi-Newton Method, DFP, BFGS [21].

I metodi gradient-based sono molto efficienti nella ricerca di minimi locali per problemi di ottimizzazione definiti da funzioni non-lineari continue e contraddistinte

da un andamento regolare. Se consideriamo ottimizzazioni discrete o problemi caratterizzati da funzioni obiettivo complesse e discontinue, è molto probabile che la ricerca si stabilizzi in un minimo locale e non globale. Per la risoluzione di questa tipologia di problemi si ricorre all'utilizzo di metodi gradient-free, ossia non basati sul calcolo del gradiente. Possiamo citare ad esempio: Random Search, Direct Search, PSO, Algoritmo Genetico [13].

1.1.2 Obiettivi della metodologia

Nonostante esista una moltitudine di algoritmi di ottimizzazione, non tutte le tipologie di problemi sono facilmente risolvibili.

É stato verificato infatti, che le soluzioni fornite da Altair Optistruct per problemi con variabili discrete sono influenzate dalla x_0 di partenza scelta dall'utente. Questo inconveniente si potrebbe risolvere utilizzando un ottimizzatore gradient-free, che sfrutta di volta in volta i risultati del software. Percorrendo questa strategia però si andrebbe incontro a tempi di calcolo molto lunghi, dovuti alla natura stocastica dell'algoritmo. Basandosi su una grande quantità di iterazioni, sarebbero necessarie centinaia di migliaia di simulazioni virtuali.

Inoltre, dalla seconda metà degli anni '90 si sta diffondendo sempre di più l'ottimizzazione multidisciplinare (MDO), in cui l'ottimo globale deve considerare aspetti relativi a discipline differenti, in termini di funzioni obiettivo e vincoli. Problemi di questo tipo non possono essere risolti con i software commerciali attualmente a disposizione, in quanto richiedono la connessione di diversi sub-modelli.

É proprio dall'esigenza di far fronte a queste due problematiche che nasce l'idea di sviluppare un codice che permetta di effettuare ottimizzazioni strutturali, indipendentemente dalla complessità della funzione da minimizzare e dalla tipologia di variabili, in tempi ridotti rispetto ad un possibile accoppiamento algoritmo gradient-free/software. L'obiettivo è, inoltre, quello di consentire l'interazione tra discipline differenti nella ricerca dell'ottimo. La metodologia sviluppata basa il proprio funzionamento sull'utilizzo dell'intelligenza artificiale, che permette di ottenere un modello surrogato in sostituzione al software CAE. Tramite tecniche di deep learning, infatti, una macchina è in grado di apprendere autonomamente determinate relazioni presenti tra input e output, restituendo risultati in tempi molto più brevi rispetto ad una simulazione virtuale eseguita per mezzo di un software. É possibile sfruttare questo vantaggio all'interno di un algoritmo genetico, ossia una metodologia di ottimizzazione gradient-free che prende ispirazione dalla selezione naturale, secondo cui, in una popolazione, solo i membri favorevoli all'adattamento riescono a sopravvivere. Grazie all'accoppiamento di questi due approcci (che verranno approfonditi nei capitoli successivi), il codice fornisce, in modo automatico, la soluzione a problemi di ottimizzazione strutturale non risolvibili (o risolvibili in tempi molto più lunghi) con metodi tradizionali o con i software commerciali attualmente a disposizione. Il beneficio è evidente soprattutto in ambito di ottimizzazioni discrete, multi-obiettivo e multidisciplinari.

1.2 Introduzione alle reti neurali

Lo sviluppo della metodologia oggetto della tesi basa le proprie fondamenta sull'utilizzo dell'intelligenza artificiale, che permette di realizzare un meta-modello predittivo che possa sostituire le analisi virtuali all'interno di un processo di ottimizzazione. Il vantaggio risiede nella rapidità con cui vengono restituiti i risultati e nella possibilità di realizzare molteplici modelli, in modo da poter considerare più discipline e più aspetti del problema contemporaneamente. Per fare ciò vengono utilizzate le reti neurali artificiali, ovvero *"modelli di calcolo matematico-informatici basati sul funzionamento delle reti neurali biologiche, ossia modelli costituiti da interconnessioni di informazioni"* [3].

Una rete neurale di fatto si presenta come un sistema "adattivo" in grado di modificare le sue interconnessioni, basandosi sia su dati esterni sia su informazioni interne, che la attraversano durante la fase di apprendimento.

Le reti del cervello umano sono la sede della nostra capacità di comprendere l'ambiente e i suoi mutamenti, e di fornire quindi risposte adattive calibrate sulle esigenze che si presentano. Tramite i sensi vengono percepiti i segnali esterni; questi vengono elaborati in informazioni attraverso un imponente numero di neuroni interconnessi tra loro in una struttura non-lineare, e variabile in base agli stimoli che riceve. É in quest'ottica che si parla dunque di modello matematico-informatico. Allo stesso modo, le reti neurali artificiali sono strutture non-lineari di dati statistici organizzate come strumenti di modellazione: ricevono segnali esterni su uno strato di nodi e tramite una fitta interconnessione di neuroni, che si attivano o inibiscono in funzione dello stimolo, restituiscono una risposta. Questo permette di far assolvere ad una macchina, in completa autonomia, ruoli che prima erano destinati esclusivamente all'essere umano [3].

1.2.1 Struttura della rete neurale

Le reti neurali sono costituite da tre strati (figura 1.2).

- **Input Layer**: è quello che ha il compito di ricevere ed elaborare i segnali in ingresso adattandoli alle richieste dei neuroni della rete. I nodi dell'input sono passivi, cioè non modificano i dati ma semplicemente duplicano il valore in ingresso e lo inviano ai nodi successivi;
- **Hidden Layer**: è quello che ha in carica il processo di elaborazione vero e proprio. Può anche essere strutturato con più livelli di neuroni;
- **Output Layer**: raccoglie i risultati dell'elaborazione degli strati precedenti e restituisce un output, che corrisponde alla previsione della variabile di risposta. Si possono avere più output.

L'unità principale di una rete neurale è il **neurone**, o nodo (figura 1.3). In ogni neurone è convogliata la somma dei valori dei nodi appartenenti al layer precedente moltiplicati per i relativi *pesi*, a cui viene aggiunto un *bias*. In questo modo si

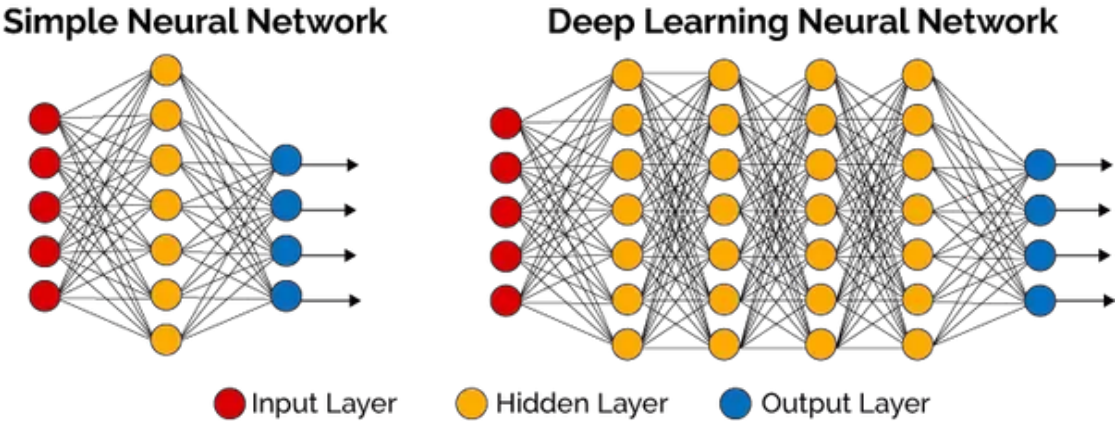

Figura 1.2: Struttura delle reti neurali

ottiene l'argomento della *funzione di attivazione*, che ha il compito di trasformare matematicamente il valore prima di passarlo al layer successivo. Questo passaggio è fondamentale per ottenere una rete neurale che sia in grado di approssimare qualsiasi tipo di funzione. Infatti, se non si utilizzasse la funzione di attivazione si otterrebbe semplicemente un modello di regressione, in grado di approssimare i dati della distribuzione con una retta. La funzione di attivazione ha quindi il compito di introdurre non-linearità [15].

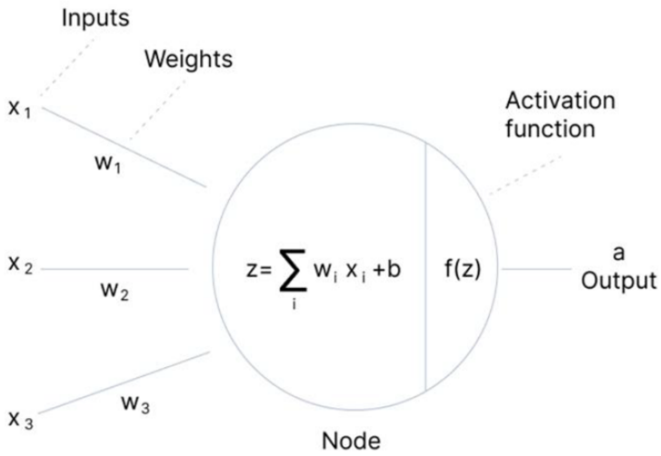

Figura 1.3: Neurone della rete neurale

1.2.2 Addestramento della rete neurale

Addestrare una rete neurale significa trovare il miglior set di pesi e bias, che garantiscano la corretta relazione tra input e output.

L'addestramento si basa sulla minimizzazione di una Loss Function e si articola nei seguenti passaggi [7] [27] (figura 1.4):

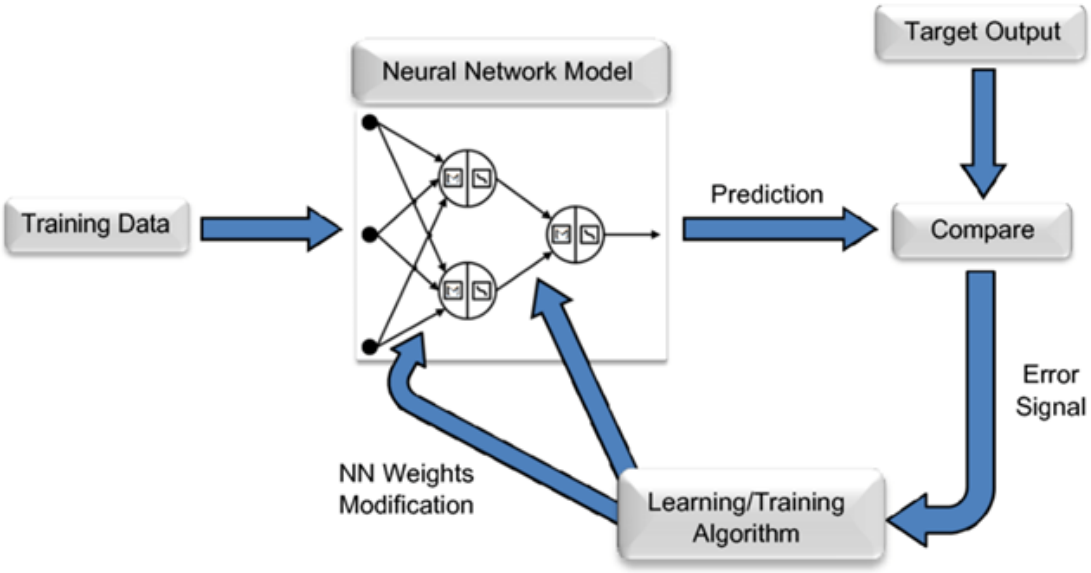

Figura 1.4: Schema addestramento rete neurale

1. Vengono inizializzati i pesi e bias della rete con valori casuali o relativi a determinate distribuzioni statistiche;
2. Viene selezionata una riga del dataset, che contiene gli input e gli output relativi ad ogni caso;
3. Forward Propagation: a partire dagli input selezionati, si ottiene l'output in base ai valori di pesi e bias;
4. Loss Function: si utilizza una funzione per misurare la differenza tra previsione e obiettivo. Lo scopo dell'addestramento è quello di minimizzare tale funzione;
5. Backward Propagation: percorrendo a ritroso la rete, vengono aggiornati i valori dei pesi e dei bias. Ciò si effettua sottraendo al valore attuale del parametro la derivata parziale della Loss Function moltiplicata per un fattore di "aggiustamento", chiamato Learning Rate. Questo fattore influenza il tempo necessario per l'addestramento e, di conseguenza, anche la qualità;

$$w_j^{i+1} = w_j^i - \frac{\partial LossFunction}{\partial w_j} \times LearningRate$$

6. Ripetendo il processo per tutte le righe del dataset, i parametri si continuano ad aggiornare;
7. Tutto il procedimento viene eseguito più volte, al fine di ottenere una rete neurale affidabile.

In base alla tipologia di addestramento scelta, i pesi e i bias vengono aggiornati dopo aver calcolato il gradiente per ogni riga oppure utilizzando la media dei gradienti di un set di righe o dell'intero dataset.

Per poter addestrare al meglio una rete neurale è richiesto lo studio accurato di una serie di parametri caratteristici, che vedremo nel dettaglio successivamente.

Capitolo 2

Problema di ottimizzazione

In fase di progettazione assume un ruolo fondamentale il processo di ottimizzazione strutturale, che permette di individuare la soluzione progettuale che fornisce le migliori prestazioni, in relazione ad un determinato obiettivo da raggiungere e a vincoli di progettazione assegnati, a partire da un determinato volume di progetto e da condizioni al contorno date (carichi, vincoli). Ad esempio, si può individuare la distribuzione di spessore di un componente stampato in plastica che permette di garantire il minor peso (obiettivo dell'ottimizzazione) nel rispetto di un requisito minimo di rigidezza alla flessione (vincolo) [26].

In particolare, esistono tre specifiche tipologie di ottimizzazione strutturale [26]:

- **Ottimizzazione Parametrica**: vengono definite a priori dall'utente alcune grandezze del modello di calcolo, alle quali viene tipicamente assegnato un range di validità tra un minimo e un massimo, su cui si può agire per raggiungere l'obiettivo. Esempi classici di parametri di ottimizzazione sono lo spessore, il modulo elastico del materiale, la rigidezza di elementi di collegamento;

- **Ottimizzazione di Forma**: si utilizzano tecniche di morphing per modificare globalmente o localmente la mesh del modello analitico al fine di migliorarne le prestazioni desiderate. Questa tecnica richiede la preparazione di un modello di calcolo di riferimento, da rendere successivamente parametrico attraverso l'applicazione del morphing.

- **Ottimizzazione Topologica**: distribuzione del materiale e definizione delle connessioni di un componente pieno/vuoto per raggiungere l'obiettivo nel rispetto dei vincoli. La principale difficoltà insita in un procedimento di ottimizzazione topologica è l'ottenimento di geometrie effettivamente realizzabili, specialmente se la tecnologica produttiva del componente è già stata fissata.

Per lo sviluppo del codice, ci si è focalizzati sull'ottimizzazione parametrica, con l'intenzione di sperimentare in futuro soluzioni applicabili anche alle altre tipologie.

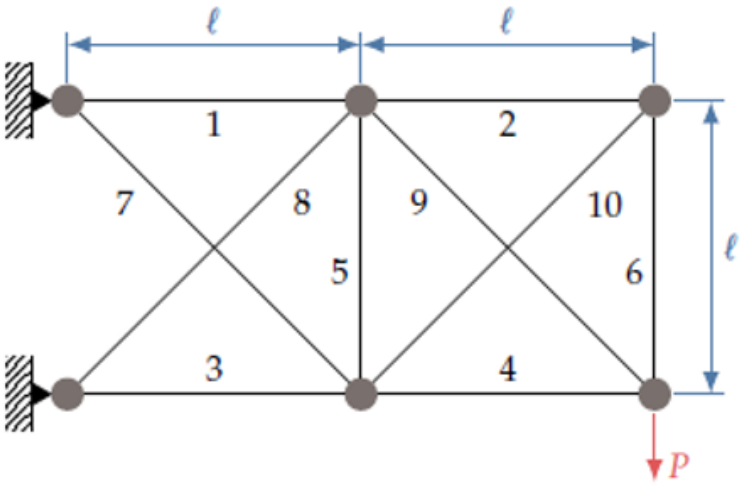

Figura 2.1: Caso di studio test

2.1 Caso di studio test

Per la realizzazione dell'infrastruttura, si è scelto di riferirsi ad un caso di studio test (figura 2.1). La prerogativa del problema è quella di poter essere risolto in Altair Optistruct, che fornisce il risultato dell'ottimizzazione con cui effettuare un confronto, per valutare l'affidabilità della metodologia. Una volta validata, potrà essere utilizzata anche per la risoluzione di problemi caratterizzati da variabili discrete e funzioni discontinue, in quanto questi dettagli non influenzano in alcun modo l'efficacia dell'algoritmo, trattandosi di una metodologia gradient-free.

Il modello in questione è una struttura reticolare costituita da 10 barre a sezione circolare, incastrata ad un'estremità e sollecitata all'estremità opposta. Lo scopo dell'ottimizzazione è la minimizzazione della massa della struttura complessiva, rispettando i limiti strutturali. Le variabili di progetto su cui poter agire sono i raggi delle 10 barre, che possono variare da 10 mm a 60 mm, in modo continuo. Per quanto riguarda i vincoli invece, essi sono imposti sullo stress assiale (trazione o compressione) di ciascuna barra, che deve rispettare il limite di snervamento del materiale. Considerando un unico materiale per tutti i segmenti della struttura, non sono necessari 10 output ma è sufficiente monitorare lo stress assiale massimo (tabella 2.1).

Il pre-processing del modello è stato realizzato in Altair Hypermesh, mentre come solutore per le analisi statiche e per l'ottimizzazione necessaria per la validazione dell'applicativo è stato utilizzato Altair Optistruct. Per quanto riguarda la tipologia di elementi, la scelta è ricaduta sulle CBEAM (elemento 1D), a cui è stata assegnata la proprietà PBEAML e il type ROD, caratteristico delle barre a sezione circolare.

VARIABILI	$\{x_i\} \quad with\, i = 1, \dots, 10$	raggi delle barre
BOUNDS	$10\ mm \leq x_i \leq 60\ mm$	i raggi delle sezioni circolari possono variare da 10 mm a 60 mm, in modo continuo
OBJECTIVE FUNCTION	$obj(x_i) = m_{TOT} = \left(\sum_i^{10} (\pi x_i^2) l_i\right) \rho$	la funzione da minimizzare corrisponde alla massa complessiva della struttura
CONSTRAINTS	$\lvert\sigma_{max}\rvert < \sigma_y$	la tensione assiale massima deve essere inferiore alla tensione di snervamento

Tabella 2.1: Sintesi del problema di ottimizzazione utilizzato come caso di studio test

Capitolo 3

Struttura del codice

La struttura del codice, che verrà illustrata in questo capitolo, è proprietà intellettuale di Capgemini Engineering.

3.1 Versione A: infrastruttura preliminare

Una volta scelto il caso di studio, si è proseguito con la scrittura degli script, prendendo spunto da «Global Optimization of a Turbine Design via Neural Networks and an Evolutionary Algorithm» [10]. Il codice è stato realizzato interamente in Python, sfruttando principalmente le seguenti librerie (collezioni di metodi e funzioni che permettono di svolgere particolari azioni):

- Pandas: utilizzata per l'analisi e la manipolazione dei database
- Keras: libreria dedicata al settaggio e all'addestramento delle reti neurali
- SciPy: contiene algoritmi e strumenti matematici

Il processo si sviluppa principalmente in 3 fasi. Per poter lavorare al meglio e testare il corretto funzionamento del codice, in un primo momento sono state realizzate separatamente:

1. Step 1: **creazione del database**
2. Step 2: **addestramento della rete neurale**
3. Step 3: **ottimizzazione**

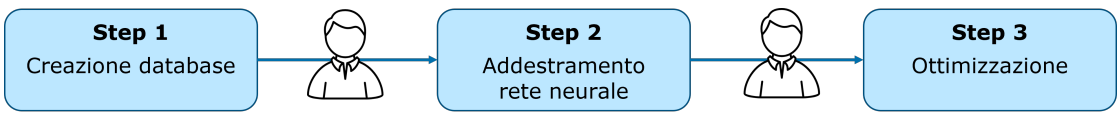

Figura 3.1: Versione A: schema dell'infrastruttura preliminare

In questo stadio è ancora necessario l'intervento dell'utente per poter muoversi da uno step all'altro, richiamando manualmente i vari file (figura 3.1). Nel corso del capitolo verrà spiegata nel dettaglio la funzione delle 3 sezioni principali del codice.

3.1.1 Step 1: creazione del database

Il database è un componente essenziale per ogni modello di intelligenza artificiale. Esso contiene informazioni sui dati, che vengono utilizzate dalla rete neurale con l'obbiettivo di trovare la corretta relazione tra input e output [25].

Nel primo step viene creato il database utilizzando delle procedure automatizzate. Considerando il caso di studio illustrato nel capitolo precedente, è possibile identificare come variabili in input i raggi delle 10 barre, mentre come unico output il massimo stress assiale. La funzione della rete è proprio quella di sostituire il software CAE nel calcolo del massimo stress assiale, in relazione alla configurazione della struttura.

Prima di passare alla generazione automatica del database, è fondamentale realizzare il modello della struttura in Altair Hypermesh (figura 3.2) e salvare la macro di lancio (.fem).

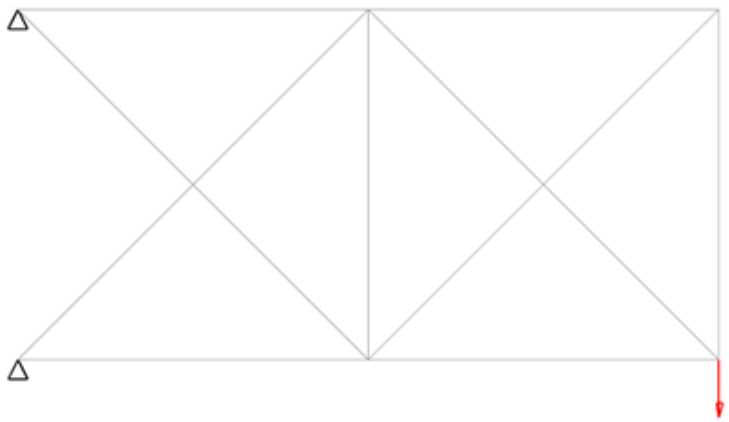

Figura 3.2: Modello in Altair Hypermesh

A questo punto, per la creazione e il salvataggio di ogni riga del database, sono stati automatizzati i seguenti passaggi:

1. Definizione di un vettore di dimensione 10. Ogni elemento corrisponde ad un raggio delle 10 barre e viene selezionato in modo randomico all'interno di un range tra 10 mm e 60 mm. Il vettore contribuisce alla generazione della matrice X del database, relativa ai dati in input;

2. Modifica del file .fem inserendo i valori dei 10 raggi definiti al punto precedente;

3. Esecuzione del run dell'analisi statica lineare in Altair Optistruct in modalità batch, cioè senza interagire in modo diretto con il software;

4. Lettura del file dei risultati .strs, da cui si ricava e si salva il massimo stress assiale. Questo valore contribuisce alla generazione della matrice Y del database, relativa ai dati in output.

Al termine di questo processo si ottiene un dataset costituito da due matrici (figura 3.3), una per i dati in input e un'altra per i dati in output, salvate come file PKL tramite un processo di serializzazione che trasforma oggetti in flusso di byte, al fine di risparmiare spazio di memoria. La matrice X ha una dimensione [n°modelli × raggi(10)], mentre la matrice Y ha una dimensione [n°modelli × max stress(1)].

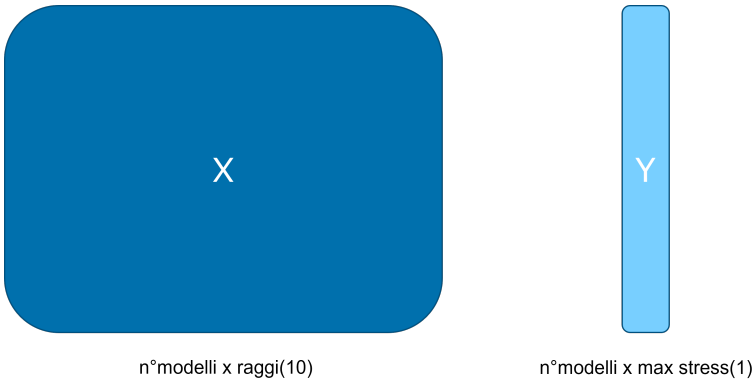

Figura 3.3: Matrici del dataset

Il numero di modelli richiesti deve essere definito inizialmente dall'utente. Il codice è in grado di effettuare un controllo sui file PKL e aggiornare il database, se esistente, senza crearne uno nuovo, garantendo un risparmio in termini di tempo. Inoltre, per evitare di bloccare il processo e perdere tutti i dati, è presente una funzione che permette di salvare il database nel momento in cui le licenze del software non siano disponibili.

3.1.2 Step 2: addestramento della rete neurale

Per la prototipazione del modello di rete neurale è stata utilizzata la libreria Keras di Python [12]. La sua realizzazione richiede di procedere secondo vari step [8], utilizzando metodi appositi presenti nella libreria (tabella 3.1).

In questa fase vengono definiti una serie di iperparametri, cioè variabili esterne al modello, il cui valore non viene determinato in base ai dati forniti in input durante il training. Vanno specificati prima di iniziare l'addestramento e, poiché non sono riassumibili da una funzione, devono essere impostati manualmente dall'utilizzatore in modo sperimentale e iterativo. La comprensione dei vari iperparametri e delle conseguenze dovute ad una loro modifica sono fondamentali per configurare al meglio il modello della rete neurale [28].

STEP	ATTIVITA'	METODO
1	caricamento del dataset	pandas.read_pickle
2	definizione della rete	*model* .add
3	configurazione della rete	*model* .compile
4	addestramento della rete	*model* .fit
5	salvataggio della rete	*model* .save
6	valutazione della rete	*model* .evaluate
7	utilizzo della rete	*model* .predict

Tabella 3.1: Step per l'addestramento della rete e relativi metodi

Per la ricerca ed il settaggio, è stata implementata una funzione che permette di effettuare in modo automatico addestramenti con configurazioni diverse e randomiche degli iperparametri. In uscita viene restituito un file di testo che associa ad ogni configurazione la qualità dell'addestramento in termini di errore compiuto nella previsione dell'output.

Caricamento dei dati

Prima di definire la struttura del modello e tutti gli iperparametri necessari, sono stati importati i file PKL contenenti le matrici X e Y del database, tramite cui verrà addestrata la rete neurale. Ciascuna matrice viene suddivisa in 3 parti, ognuna delle quali ha un compito specifico.

- Training Set - 60%: questi dati vengono utilizzati esclusivamente in fase di addestramento, durante il quale il modello impara la relazione tra le X e le Y;
- Validation Set - 20%: è un set di dati utilizzato per effettuare una valutazione della rete durante il settaggio dei vari iperparametri. Per queste analisi non ci si basa sul Training Set perché si vuole verificare l'affidabilità della rete nei confronti di dati che non conosce;
- Test Set - 20%: una volta che la rete restituisce buoni risultati su Training Set e Validation Set, il modello può essere testato su nuovi dati per una validazione finale.

Per la scelta dei rapporti in percentuale tra i tre Set si è preso spunto da informazioni presenti in articoli scientifici [23] [16], e successivamente personalizzate.

Prima di utilizzare il dataset per l'addestramento della rete, viene effettuato un pre-processing sui dati in input. Questo passaggio ha lo scopo di normalizzare tra 0 e 1 i valori della matrice X, in quanto le reti neurali apprendono efficacemente se lavorano con numeri piccoli [19].

É stato utilizzato un metodo presente in una libreria di Python, chiamato *MinMaxScaler*, che permette di definire inizialmente i parametri di scaling sulla

base dei dati a disposizione e successivamente applicare lo *scaler*, che viene poi salvato (.save), dovendo essere utilizzato sempre in accoppiamento con la rete neurale.

Definizione della rete

PARAMETRO	FUNZIONE
units	numero di neuroni del layer
activation	funzione di attivazione, che ha il compito di applicare una trasformazione matematica alla somma pesata degli input per introdurre non-linearità nella rete
kernel_initializer	utilizzato per inizializzare la matrice dei pesi
bias_initializer	utilizzato per inizializzare i bias
kernel_regularizer	aggiunge una penalità alla Loss Function per ridurre il valore dei pesi ed evitare quindi overfitting, cioè il sovradattamento rispetto ai dati del training
activity_regularizer	aggiunge una penalità all'output del neurone, agendo sui pesi e bias
dropout	ha la funzione di disattivare alcuni neuroni del layer, al fine di generalizzare l'addestramento della rete ed impedire un eccessivo adattamento

Tabella 3.2: Definizione della rete - metodo *add*

Per la costruzione della rete viene utilizzato un modello "Sequential", generato tramite aggiunta di Dense Layer successivi. Un Dense Layer è costituto da un insieme di neuroni, che ricevono gli output di ogni nodo dello strato precedente e operano tramite prodotto matriciale [29]. É necessario quindi definire inizialmente il numero di layers della rete e la loro dimensione. Per ognuno di essi vanno poi settati una serie di iperparametri (tabella 3.2).

Struttura della rete Nella realizzazione di un modello di rete neurale è di fondamentale importanza la scelta della sua struttura, cioè il numero di Hidden Layers e la loro dimensione (numero di neuroni).

Per fare ciò, è stata utilizzata una metodologia che permettesse di generare diversi modelli con numero e dimensione dei layer randomici. Implementando una funzione che restituisse l'errore commesso da ogni modello nella predizione dell'output, è stato possibile valutare le diverse opzioni e capire in che modo i parametri influenzassero l'addestramento.

Dall'analisi dei risultati relativi alle varie tipologie di strutture, è emerso che risulta vantaggioso utilizzare un numero di neuroni nei primi strati maggiore rispetto al numero degli input. Incrementando la dimensione della matrice dei pesi, aumenta la possibilità di trovare una combinazione di pesi e bias che conduca ad un addestramento corretto.

Come illustrato nello schema di figura 3.4, la struttura del modello realizzato è costituita da 4 Hidden Layers, delle seguenti dimensioni: **160-80-40-20**. Essendo sufficiente il solo massimo stress assiale, l'Output Layer è composto da un unico nodo. Il numero complessivo di parametri da aggiornare durante l'addestramento è pari a 18721, più precisamente 18420 pesi e 301 bias. Il calcolo è presente nella tabella 3.3.

	Input Layer	Hidden Layer 1	Hidden Layer 2	Hidden Layer 3	Hidden Layer 4	Output Layer	Totale parametri
numero di neuroni	10	160	80	40	20	1	
pesi	0	160x10 = 1600	80x160 = 12800	40x80 = 3200	20x40 = 800	1x20 = 20	18420
bias	0	160	80	40	20	1	301

Tabella 3.3: Parametri da aggiornare durante l'addestramento

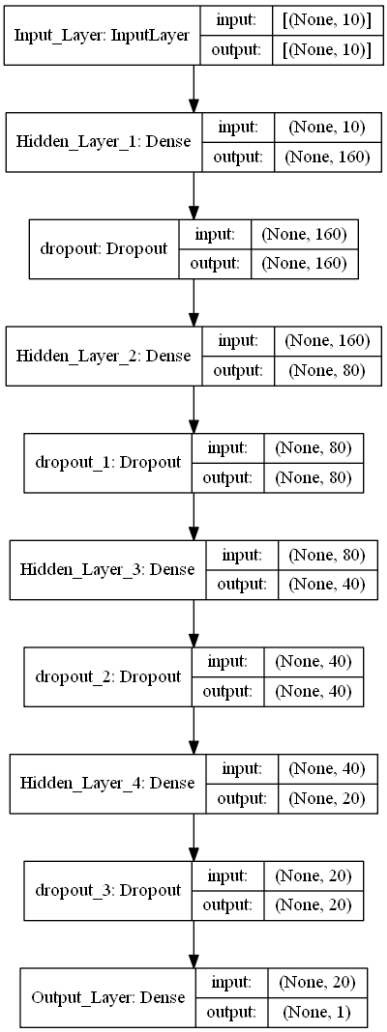

Figura 3.4: Struttura della rete neurale

Activation Function É una funzione che ha il compito di applicare una trasformazione matematica alla somma pesata degli input che convogliano in ogni nodo. Dopo uno studio accurato delle varie possibilità [2], è stata selezionata una funzione che effettui una trasformazione non-lineare, per i seguenti motivi:

- permette la Backward Propagation, avendo la derivata variabile in base all'input e non costante. É quindi possibile ripercorrere la rete neurale al contrario e ottimizzare i pesi;
- i layers multipli assumono senso perché ogni input subisce una trasformazione.

Essendo quello trattato un problema di regressione, in cui in uscita è richiesto un valore continuo (massimo stress assiale) e non la probabilità di appartenere ad una certa categoria, come nel caso dei problemi di classificazione, per tutti gli Hidden Layers la scelta della funzione di attivazione è ricaduta sulla **ReLU**: Rectified Linear Unit (figura 3.5).

Figura 3.5: ReLU - Rectified Linear Unit

Nonostante possa sembrare una funzione lineare, ReLU è derivabile e permette quindi la Backward Propagation; allo stesso tempo è efficiente dal punto di vista computazionale.

I principali vantaggi comportati dall'utilizzo di questa funzione sono:

- maggiore efficienza perché attiva solo un certo numero di neuroni;
- accelera la discesa del gradiente verso il minimo globale della funzione di costo grazie alla sua linearità.

Lo svantaggio è dovuto al fatto che la parte negativa del grafico produce un valore del gradiente nullo, che comporta il mancato aggiornamento di alcuni pesi e bias durante la Backward Propagation. Di conseguenza alcuni neuroni potrebbero non attivarsi mai.

Per evitare il problema della "Neuron Death" sono stati fatti alcuni tentativi con la funzione chiamata Leaky ReLU, che permette di scegliere il valore del coefficiente angolare per $x<0$. Nonostante la grande quantità di scelte differenti, non si è ottenuto alcun miglioramento dei risultati.

Per quanto riguarda l'Output Layer, è stata utilizzata una funzione di attivazione puramente lineare.

Kernel Initializer É possibile scegliere una distribuzione per inizializzare i pesi della rete neurale. Non conviene assegnare un valore costante a tutti i pesi perché in questo modo tutti i neuroni restituirebbero gli stessi risultati e gli stessi

gradienti, causando un addestramento simmetrico. Inizializzare i pesi con valori troppo bassi potrebbe comportare un addestramento troppo lento, mentre assegnare valori troppo alti causerebbe divergenza. Spesso i pesi vengono assegnati secondo una distribuzione normale. Utilizzando questa metodologia ed effettuando più addestramenti, venivano restituiti risultati molto variabili. Per tale motivo si è deciso di utilizzare **HeUniform**, consigliato appunto nei casi in cui la Activation Function sia la ReLU. In questo modo i pesi vengono inizializzati secondo una distribuzione uniforme tra [-*limit*, *limit*], in cui $limit = \sqrt{\frac{6}{fan\_in}}$. Nella formula *fan_in* è il numero di unità in ingresso nella matrice dei pesi.

Bias Initializer Per quanto riguarda l'inizializzazione dei valori dei bias, solitamente viene impostata a zero perché la simmetria nell'addestramento viene già rotta con l'utilizzo del kernel_initializer. In questo caso però si è preferito ricorrere all'utilizzo di **HeUniform** perché garantisce un miglioramento dei risultati.

Regolarizzazione L'utilizzo di layers di dimensioni abbastanza grandi potrebbe causare un eccessivo adattamento della rete neurale ai valori utilizzati per l'addestramento, a discapito di valori differenti ancora sconosciuti. Ciò si rifletterebbe in una grande discrepanza tra l'errore relativo ai dati del Training Set e quello relativo al Validation Set. Per ridurre questa differenza, è possibile introdurre delle penalità alla Loss Function e agli output dei neuroni tramite i Regularizer [20]. Un'altra metodologia potrebbe essere la disattivazione di una certa percentuale di neuroni di alcuni layers per mezzo del Dropout [4].

- Regularizer: è necessario settare il fattore di regolarizzazione che, moltiplicato per i pesi, aggiunge una penalità alla Loss Function. É possibile scegliere tra la regolarizzazione L2, che contempla i quadrati dei pesi e fa in modo che tutti gli input abbiano la stessa importanza, oppure la L1, che invece non utilizza l'elevamento a potenza e permette ad alcuni input di prevalere su altri;

- Dropout: è necessario settare per ogni Hidden Layer la percentuale di neuroni che si vogliono disattivare, solo durante il training. Ad ogni iterazione, i nodi che vengono disattivati sono scelti casualmente.

Per quanto riguarda il modello di rete in esame, non è stato necessario nessun intervento in termini di regolarizzazione, in quanto riesce a predire con una qualità molto simile sia gli output del Training Set che quelli del Validation Set. Quindi sia il Dropout che i Regularizer sono settati al valore 0.

Configurazione della rete

Questa sezione è dedicata alla configurazione del modello per l'addestramento (tabella 3.4).

Per comprendere meglio ciò che verrà spiegato successivamente, è importante definire la Learning Curve (figura 3.6). Questa curva rappresenta l'andamento della

PARAMETRO	FUNZIONE
loss	funzione da minimizzare per l'aggiornamento dei pesi e dei bias
optimizer	algoritmo utilizzato per minimizzare la funzione di costo, al fine di aggiornare gli attributi della rete (pesi e bias) al fine di ridurre l'errore

Tabella 3.4: Configurazione della rete - metodo *compile*

Loss Function (funzione che misura l'errore tra obiettivo e previsione) relativa al Training Set e al Validation Set, durante l'addestramento della rete neurale.

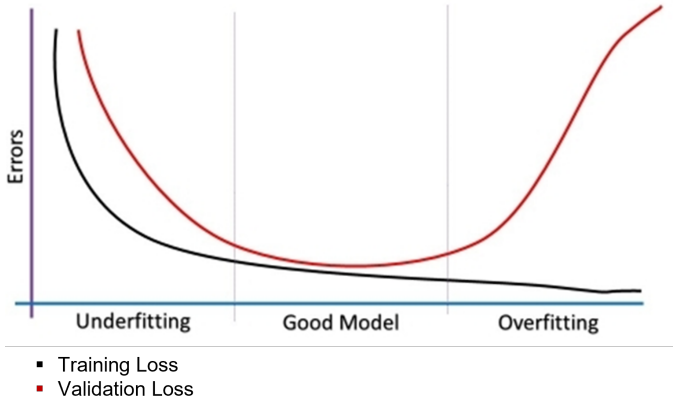

Figura 3.6: Learning Curve

L'obbiettivo è quello di avere le due curve più sovrapposte possibile, ad indicare che il modello si comporta allo stesso modo sia con i dati utilizzati per l'addestramento sia con quelli necessari per la Validation. Se le curve sono distanti tra di loro o presentano molte oscillazioni, può significare:

- addestramento del modello errato (underfitting o overfitting);
- i due Set non rappresentano fedelmente l'intero campo dei dati, quindi le due curve potrebbero riferirsi a distribuzioni diverse;
- la quantità di dati utilizzati per l'addestramento non è sufficiente;
- uno dei due Set è più facile da predire rispetto all'altro.

Se l'addestramento della rete dovesse avvenire in modo errato, si potrebbe ricadere in due casistiche (figura 3.7):

- Validation Loss » Training Loss ⇒ OVERFITTING: la rete si comporta bene solo con i dati utilizzati per l'addestramento e non è in grado di generalizzare. É normale che si verifichi un po' di overfitting;
- Validation Loss « Training Loss ⇒ UNDERFITTING: la rete non riesce a predire gli output corretti.

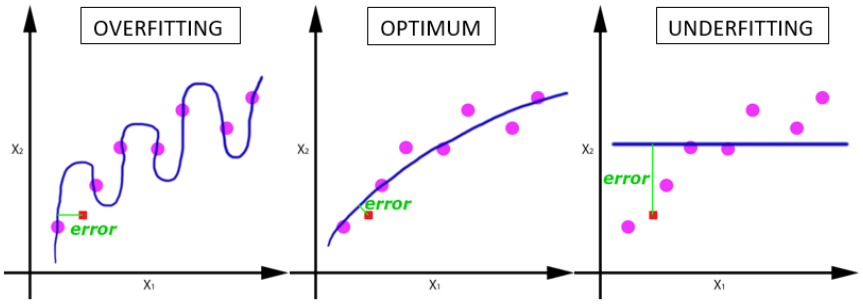

Figura 3.7: Overfitting e Underfitting

Per una corretta lettura dei grafici, viene anticipato un iperparametro che verrà approfondito in seguito. Le Epochs corrispondono al numero di volte che il dataset viene presentato alla rete neurale per effettuare più passaggi di Forward/Backward Propagation su tutto il campo di dati, al fine di ottenere un addestramento soddisfacente.

Loss La Loss Function è la funzione da minimizzare per ottenere il corretto valore degli attributi della rete neurale (pesi e bias). La scelta di tale funzione è dipendente dalla tipologia del problema [6].

Per problemi di regressione, quando cioè si è interessati al valore dell'output in un intervallo continuo, è necessario monitorare l'errore che viene compiuto dalla previsione rispetto all'obiettivo. Quando si tratta di classificazione, invece viene valutata l'accuratezza, ossia la percentuale con cui agli input viene associata la classe di riferimento corretta.

Quando il target è ampio e si lavora con grandi valori è consigliato l'utilizzo del MSLE (Mean Squared Logarithmic Error). Viene prima calcolato il logaritmo naturale e poi la media quadratica, in modo da non punire troppo pesantemente il modello. Con questo metodo, nel corso dell'addestramento la Loss Function subisce un'importante riduzione; l'errore medio tra valori previsti e valori obiettivo è comunque abbastanza alto. Probabilmente il logaritmo riduce talmente tanto la funzione di costo da compromettere l'aggiornamento degli attributi della rete. Si è preferito quindi riferirsi a funzioni di costo meno complesse, come MAE (Mean Squared Error) e MAPE (Mean Absolute Percentage Error). La prima è relativa all'errore medio assoluto, mentre la seconda all'errore medio percentuale. Si è optato per il **MAPE**, che fornisce un'informazione più precisa sull'effettiva qualità della rete neurale addestrata, in relazione alla grandezza da valutare.

$$MAPE = \frac{100}{n} \sum_{j=1}^{N} \left| \frac{A_j - P_j}{A_j} \right| \quad \text{A: valore atteso P: valore previsto} \tag{3.1}$$

Nei grafici presenti nelle pagine successive, la Loss corrisponde sempre all'errore medio percentuale.

Optimizer Per la scelta di questo parametro ci si è affidati alle indicazioni trovate in bibliografia [9] [5], confermate tramite addestramenti in cui sono stati utilizzati diversi algoritmi di ottimizzazione. L'algoritmo **Adam** (Adaptive Moment Estimation) infatti, presenta i seguenti vantaggi:

- semplice da implementare, considerando che i valori di alcuni parametri sono già impostati di default;
- efficiente computazionalmente;
- richiede poca memoria;
- adatto a problemi che presentano grandi quantità di dati e parametri;
- appropriato per problemi con variazioni del gradiente.

Adam combina i vantaggi di altri due algoritmi.

- AdaGrad: caratterizzato da un Learning Rate adattativo, in base alla variabilità del peso da aggiornare. Se un parametro si aggiorna troppo velocemente, viene ridotto il relativo Learning Rate, e viceversa;
- RMSProp: utilizzando il momentum, viene accelerata la convergenza lungo le direzioni rilevanti e vengono ridotte le fluttuazioni lungo direzioni irrilevanti. Il momentum considera l'andamento dei gradienti calcolati in precedenza.

Addestrando la rete anche con i due algoritmi sopracitati, si è verificato che l'utilizzo di Adam garantisce risultati più soddisfacenti.

Per quanto riguarda la maggior parte dei parametri che caratterizzano l'ottimizzatore Adam, è stato lasciato il valore di default. Quindi è stato necessario impostare solamente il Learning Rate, che definisce l'intensità con cui aggiornare gli attributi caratteristici della rete, cioè pesi e bias. Come mostrato nel grafico in figura 3.8, questo parametro ha una grande influenza sull'esito dell'addestramento [11].

Per valori troppo bassi, richiede molto tempo prima di raggiungere l'ottimo. Per valori troppo alti l'addestramento è più rapido, ma si rischia di stazionare in un sub-ottimo e non giungere a convergenza.

Osserviamo i grafici (figura 3.9) che mostrano l'andamento della Loss Function durante tre addestramenti caratterizzati da valori di Learning Rate differenti. Si nota che un Learnging Rate troppo alto non consente agli attributi della rete di aggiornarsi nel modo corretto e di conseguenza non si ottiene alcun miglioramento in termini di errore. Riducendo il valore di questo iperparametro è possibile ottenere un addestramento contraddistinto da curve di costo con meno fluttuazioni, a discapito però del tempo richiesto. Si è scelto quindi di impostare il Learning Rate pari a **0.01** per avere un compromesso tra qualità e tempistiche dell'addestramento.

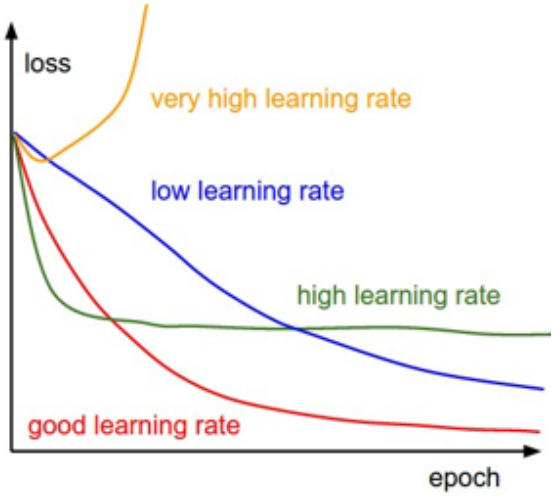

Figura 3.8: Influenza del Learning Rate sull'esito dell'addestramento

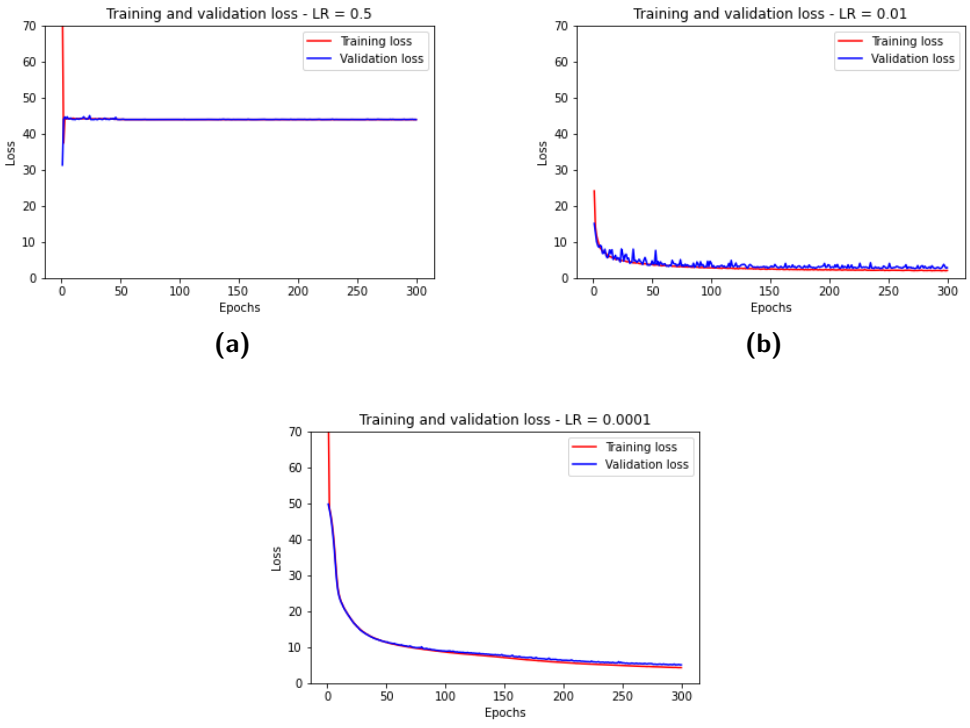

Figura 3.9: Confronto tra addestramenti con LearningRate differenti

Addestramento della rete

Questa è l'ultima sezione che richiede l'impostazione di alcuni iperparametri (tabella 3.5).

PARAMETRO	FUNZIONE
epochs	definisce il numero di volte che l'algoritmo di apprendimento lavora sull'intero database. All'interno di ogni epoca vengono fatte tante iterations quanti sono i batch
batch_size	definisce il numero di righe (samples) del database da elaborare insieme prima di comparare le previsioni con gli obbiettivi e calcolare l'errore, per poi aggiornare pesi e bias

Tabella 3.5: Addestramento della rete - metodo *fit*

Epochs Il numero di Epochs definisce il numero di volte in cui deve essere fornito alla rete neurale il dataset ed è uno degli iperparametri più importanti. Man mano che le Epochs passano, i pesi vengono aggiornati e si ottengono miglioramenti della curva di costo [24].

Solitamente più il dataset è grande e disomogeneo, più Epochs sono necessarie affinché si ottenga una rete addestrata affidabile. Un numero ridotto infatti potrebbe causare underfitting, ma allo stesso tempo non è consigliato usarne troppe perché si rischia di cadere nell'overfitting.

Dall'analisi dei grafici in figura 3.10 è possibile notare l'influenza del numero di Epochs sull'addestramento. L'utilizzo di poche Epochs non permette alla Loss Function di stabilizzarsi; ciò significa che la rete neurale non è ancora generalizzata, quindi si comporta bene con determinati input e male con altri. Quando invece sono troppe, dopo un certo istante la funzione di costo non subisce più alcun miglioramento e si rischia di addestrare eccessivamente la rete. Infatti, nel grafico (c) la distanza tra Validation Loss e Training Loss è leggermente maggiore rispetto a quella nel grafico (b); è sintomo di overfitting. Si è ipotizzato che 500 Epochs siano sufficienti per raggiungere ottimi risultati. A causa della randomicità dell'addestramento della rete neurale, dovuta a Initializer e aggiornamento degli attributi caratteristici tramite il calcolo del gradiente, non è richiesto sempre lo stesso tempo per il raggiungimento della convergenza. Per questo motivo si è deciso di considerare un margine di sicurezza, impostando **1000** Epochs, ed utilizzare una funzione (spiegata successivamente nel dettaglio) per arrestare l'addestramento nel momento opportuno. La scelta è stata influenzata anche dal fatto che aumentando il numero di modelli del database sono necessari ancor più passaggi della rete sui dati.

Batch Size Quando il dataset è di dimensioni notevoli, anche con elaboratori molto performanti non è possibile processare tutte le righe in contemporanea. É possibile suddividere i dati del Training in sottoinsiemi di dimensione prefissata, chiamati Batch [24]. In questo modo diminuisce notevolmente l'utilizzo della memoria centrale nella fase dell'addestramento, ma allo stesso tempo la stima del gradiente, e quindi anche la minimizzazione della Loss Function, potrebbero essere poco accurate. Infatti, utilizzando valori piccoli di Batch Size, la selezione randomica di gruppi di dimensioni limitate conduce di volta in volta ad errori differenti.

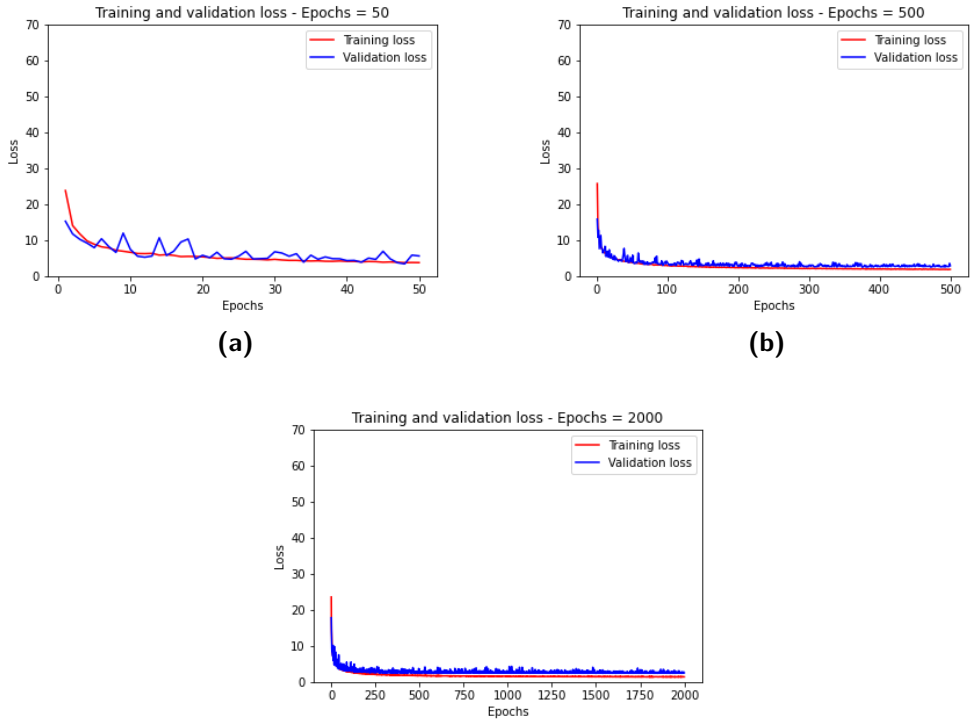

Figura 3.10: Confronto tra addestramenti con Epochs differenti

In base alla dimensione dei Batch si ottengono diverse tipologie di addestramento [18] (tabella 3.6):

- Batch Gradient Descent: viene utilizzata la media dei gradienti di ogni riga del dataset e di conseguenza si ottiene una curva con meno oscillazioni. (Iterazioni=1)

- Stochastic Gradient Descent: i pesi vengono aggiornati dopo aver calcolato il gradiente per ogni riga, comportando un costo computazionale ridotto. (Iterazioni=samples)

- MiniBatch Gradient Descent: per aggiornare i pesi viene utilizzata la media dei gradienti delle righe contenute in ogni Batch. É una via di mezzo tra i due metodi citati in precedenza. (Iterazioni=samples/BatchSize)

Per la valutazione di questo parametro è stata utilizzata una funzione che permette di effettuare vari addestramenti con Batch Size differenti e comparare i risultati ottenuti.

A valle di diversi tentativi, si è scelto di utilizzare la metodologia MiniBatch Gradient Descent, con una dimensione dei Batch pari a **16**, in modo tale da avere un compromesso tra qualità dell'addestramento della rete neurale e tempo necessario.

	BatchSize	Loss Function	Costo computazionale	Tempo	Consigliato per grandi DataSet	Soluzione	Uscire dai minimi locali
Stochastic Gradient Descent	1	molte variazioni	basso	basso	si	buona	si
MiniBatch Gradient Descent	1<BS<samples	un pò più regolare di SGD	medio	medio			
Batch Gradient Descent	samples	regolare	alto	alto	no	ottima	difficilmente

Tabella 3.6: Confronto tra diverse tipologie di addestramento in relazione al Batch Size

Funzioni di callback Una callback è un oggetto che permette di effettuare azioni durante l'addestramento della rete neurale. É possibile quindi impostare una lista di callbacks all'interno del metodo *fit*, che hanno la funzione di monitorare alcuni parametri ed agire di conseguenza.

Considerando che il Learning Rate e il numero di Epochs sono gli iperparametri più influenti nell'addestramento e allo stesso tempo i più difficili da impostare, sono state utilizzate due callbacks per la loro definizione.

- **ReduceLROnPlateau**: permette di ridurre il Learning Rate durante l'addestramento nel momento in cui la funzione di costo si stabilizza;

- **EarlyStopping**: è stata utilizzata per evitare che un numero troppo alto di Epochs causi overfitting. Grazie a questa funzione è stato possibile impostare questo iperparametro maggiore di 500, con la certezza che l'addestramento venga interrotto se non dovessero verificarsi miglioramenti della Loss Function.

Dovendo effettuare un monitoraggio, per l'utilizzo di queste funzioni è richiesto il settaggio di alcuni parametri:

- *monitor*: è la variabile da monitorare. Si è scelto di valutare la Validation Loss e non la Training Loss perché l'obiettivo è quello di creare un modello che sia in grado di generalizzare su tutti i dati. Controllando la Training Loss non è possibile capire se l'addestramento si stia svolgendo correttamente o se la rete stia semplicemente imparando a memoria le relazioni tra input e output del Training Set.

- *min_delta*: è la minima variazione della variabile monitorata tale da essere considerata un miglioramento. É stata impostata a 0.01.

- *patience*: corrisponde al numero di Epochs senza miglioramenti dopo le quali interviene la callback. É stato impostato a 30 per il ReduceLROnPlateau e a 60 per l'EarlyStopping, dando la possibilità di verificare se con la riduzione del Learning Rate si possa ottenere qualche miglioramento.

Mentre i parametri appena descritti sono fondamentali per entrambe le callbacks utilizzate, ce ne sono altri specifici della funzione. Per quanto riguarda il ReduceLROnPlateau bisogna definire l'entità del riduttore (0.75); nella definizione dell'EarlyStopping invece bisogna scegliere se salvare la configurazione di pesi e bias corrispondente al miglior risultato in termini di Loss.

Al termine dell'addestramento, il metodo *fit* restituisce un oggetto History. Il suo attributo *history* è un dizionario che contiene l'andamento di Loss e Metrics (funzioni utilizzate per valutare il modello) per ogni Epoch, relativo a Training e Validation.

Salvataggio della rete

Al termine dell'addestramento la rete neurale viene salvata con il metodo *save*, in formato .h5. Per tenere traccia dei vari addestramenti, all'interno della cartella le varie reti sono nominate in base alla data e all'ora in cui sono stati effettuati.

Valutazione della rete

Per valutare la rete si utilizza il metodo di Keras *evaluate*, che restituisce il valore della Loss e può essere applicato ad ognuno dei 3 Set. Essendo una validazione finale il valore di interesse è il MAPE relativo al Test Set, con cui si verifica se la rete è generalizzata anche su dati che non conosce.

Per avere una visione d'insieme è stato implementato anche uno script che calcola l'errore medio e l'errore massimo assoluti, associati ad ogni Set.

Utilizzo della rete

Per poter utilizzare la rete bisogna effettuare alcuni passaggi, spiegati di seguito:

1. importare lo *scaler* utilizzato per normalizzare il dataset prima dell'addestramento;
2. importare il modello della rete, salvato in formato .h5;
3. definire un array che abbia una dimensione pari al numero di input della rete neurale e scalarlo;
4. utilizzare il metodo *predict* per ottenere l'output.

3.1.3 Step 3: ottimizzazione

Una volta addestrata la rete neurale viene effettuata l'ottimizzazione, utilizzando l'Algoritmo Genetico. É un metodo gradient-free, ossia che non sfrutta il calcolo del gradiente per la ricerca del minimo, ed è consigliato per la risoluzione di problemi di ottimizzazione caratterizzati da variabili discrete e da funzioni discontinue che presentano minimi locali, per i quali risulta inefficace l'impiego di algoritmi gradient-based.

L'Algoritmo Genetico si basa sul principio della selezione naturale e sfrutta criteri simili a quelli con cui la genetica spiega l'evoluzione della specie. Le possibili soluzioni di un problema vengono dette "individui" e contribuiscono a formare la "popolazione". A partire dalla prima generazione, che può essere inizializzata in modo randomico o con metodologie statistiche, tramite la funzione di *fitness* viene

calcolata la probabilità di riproduzione di ciascun individuo della popolazione in relazione al problema. A questo punto viene effettuato il cross-over, ovvero la combinazione delle soluzioni per la formazione della nuova generazione. Solitamente vengono considerati principalmente gli individui con un valore maggiore della funzione di *fitness*, ma non sono comunque da escludere a priori le soluzioni caratterizzate da poca attendibilità perché attraverso l'evoluzione potrebbero generare degli elementi figli validi. Oltre al cross-over, l'algoritmo attua delle variazioni casuali all'interno delle soluzioni, chiamate mutazioni, al fine di ottenere una maggiore varietà di individui all'interno della popolazione. L'ottimizzazione termina quando la totalità della popolazione giunge a convergenza[1].

Per la ricerca dell'ottimo è stata utilizzata la funzione *differential_evolution* della libreria di Python SciPy[22], che sfrutta la rete neurale addestrata per il calcolo del massimo stress assiale, su cui è imposto il vincolo strutturale del problema. Lo step 3 è strutturato secondo i seguenti passaggi:

1. caricamento dello scaler (.save) e del modello della rete neurale addestrata (.h5);
2. impostazione della Objective Function, che corrisponde alla massa totale della struttura;
3. definizione della funzione che permette di calcolare il massimo stress assiale, tramite il metodo *predict*, specifico delle reti neurali. É fondamentale trasformare la soluzione candidata in array, per poi essere normalizzato prima di interrogare la rete;
4. impostazione dei Constraints, che viene fatta richiamando la funzione definita in precedenza e settando i limiti inferiore e superiore pari a [0 , 289.6];
5. impostazione dei Bounds, che in questo caso sono uguali per tutte e 10 le variabili [10mm , 60mm];
6. risoluzione del problema di ottimizzazione con la funzione *differential_evolution*;
7. valutazione e salvataggio dell'ottimo trovato, sia in termini di massa che di configurazione dei 10 raggi.

3.1.4 Analisi dei risultati

Per validare l'efficacia della metodologia, viene effettuata un'analisi dei risultati, prima sullo step 2 e successivamente sullo step 3.

Utilizzando un maggior numero di modelli per la creazione del database, si permette alla rete di esplorare più casistiche e quindi di giungere ad un miglior addestramento. Per questo motivo sono stati eseguiti una serie di addestramenti al variare della dimensione del database. L'obiettivo è quello di ridurre l'errore compiuto dalla rete nella previsione degli output, incrementando il numero di modelli.

Dal grafico (figura 3.11) si nota che per dimensioni del database relativamente piccole si ottiene un importante miglioramento. Dopo un certo numero di modelli, la rete necessita di incrementi sempre più grandi per poter restituire risultati migliori. Osservando l'andamento della funzione, si è preferito non andare oltre i 14000 modelli perché il tempo richiesto per la creazione di ulteriori righe del database sarebbe stato eccessivo rispetto al possibile guadagno ottenuto sull'errore. Al termine dell'addestramento, la rete neurale risulta affetta da un errore medio non trascurabile, pari a circa il 2,5%. L'errore è relativo all'output richiesto e cioè al massimo stress assiale.

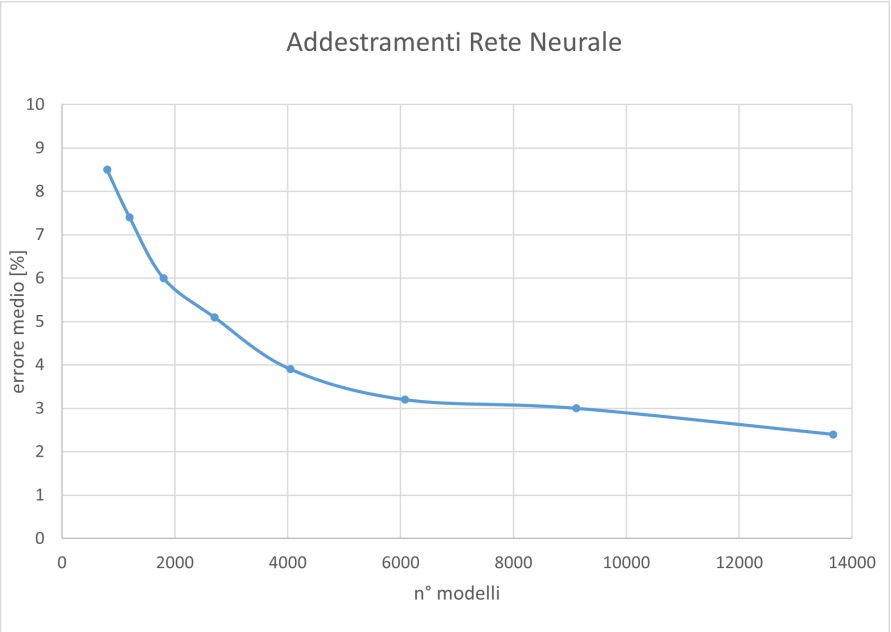

Figura 3.11: Analisi degli addestramenti della rete neurale - Versione A

Per la validazione del risultato finale invece, è possibile confrontare i valori forniti dall'infrastruttura con quelli proposti dall'ottimizzatore di Altair Optistruct. Per avere una visione completa, viene analizzato il valore dei 10 raggi e il valore della massa complessiva, associata alla configurazione ottimale della struttura (tabella 3.7). Si nota che per la maggior parte dei valori dell'array soluzione, viene compiuto un errore elevato. La causa è riconducibile alla poca affidabilità della rete neurale che, restituendo dei valori di stress distanti da quelli reali, fornisce all'Algoritmo Genetico indicazioni fuorvianti sul rispetto dei limiti strutturali. Questa problematica comporta un errore del 14,9% sulla massa, che conduce ad un'ottimizzazione deludente rispetto ai risultati attesi.

		Optistruct	Infrastruttura	errore [%]
variabili [mm]	x1	38,6	38,4	-0,5
	x2	10,0	10,0	0,0
	x3	27,6	30,0	8,7
	x4	23,0	21,8	-5,2
	x5	10,0	11,7	17,0
	x6	10,0	12,3	23,0
	x7	11,0	15,6	41,8
	x8	27,1	30,0	10,7
	x9	26,9	27,1	0,7
	x10	10,0	16,3	63,0
objective function [kg]	massa	47,66	54,75	14,88

err < 1%	1% ≤ err < 3%	err ≥ 3%

Tabella 3.7: Confronto tra risultati di Optistruct e quelli dell'infrastruttura - Versione A

3.2 Versione B: INNO - Iterative Neural Network Optimization

Avendo riscontrato errori importanti sui valori da ottimizzare, si è scelto di implementare due loop all'interno del processo per il controllo di determinati parametri d'interesse, al fine di giungere alla soluzione tramite iterazione successive. Al contempo è stato automatizzato il passaggio di informazioni tra i 3 step, realizzando un applicativo che richiede l'intervento dell'utente solo per il settaggio iniziale. É così che nasce INNO: un codice che esegue processi di ottimizzazione iterativi sfruttando le reti neurali, in modo totalmente automatico. Di seguito verrà illustrata nel dettaglio la logica di funzionamento dei due loop.

3.2.1 Loop RETRAIN

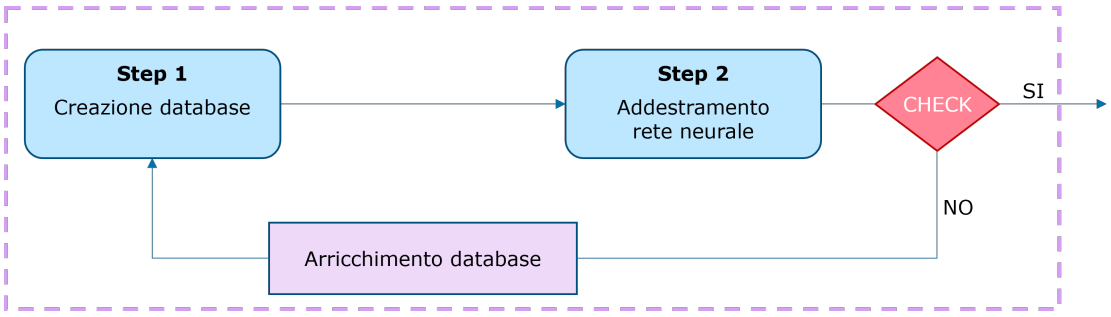

Figura 3.12: Loop RETRAIN [step1-step2]

Il primo ciclo viene effettuato tra lo step 1 e lo step 2 (figura 3.12), con l'obiettivo di monitorare l'esito dell'addestramento al variare del numero di modelli utilizzati per la costruzione del database. In questo modo non si rischia di effettuare troppe simulazioni senza ottenere ad alcun miglioramento, o di farne poche e quindi precludersi di avere una rete neurale addestrata meglio. Inizialmente è richiesta l'impostazione di tre parametri: il numero di modelli richiesti inizialmente, la percentuale di incremento al termine di ogni addestramento, la "pendenza" minima della curva che mette in relazione l'errore medio assoluto con il numero di modelli. Quest'ultimo parametro viene valutato secondo la formula seguente:

$$m = \frac{err_{j-2} - err_j}{(n\_modelli_j - n\_modelli_{j-2})/1000} \tag{3.2}$$

Viene calcolato il delta tra l'addestramento j e l'addestramento *j-2* per evitare di fermare il loop prematuramente.

Il loop è articolato nelle seguenti fasi:

1. Generazione di un numero di modelli iniziale per la creazione del database;
2. Addestramento della rete neurale;
3. Salvataggio delle informazioni utili su un file di log (numero di modelli, errore medio percentuale e assoluto, percentuale di righe del Test Set affette da un errore maggiore di un certo valore, tempo richiesto per le simulazioni dei modelli, durata dell'addestramento, pendenza della funzione da monitorare, nome della rete neurale) (figura 3.13);
4. Controllo sul valore della pendenza. In caso di esito negativo, vengono effettuate ulteriori simulazioni per arricchire il database e si ritorna al punto 2; altrimenti si procede con l'ottimizzazione.

n° modelli	err_medio[Mpa]	err_medio[%]	err>15MPa[%]	err>5MPa[%]	tempo step1	tempo step2	pendenza	nome NN
800	48.5	8.5	66.5	88.0	1h 3m	35s		NN_04_06_2022__16_08_59
1200	42.4	7.4	62.7	87.0	28m	59s		NN_04_06_2022__16_09_45
1800	33.9	6.0	54.2	81.6	47m	1m 0s	11.7	NN_04_06_2022__16_10_28
2700	27.7	5.1	54.8	81.2	1h 4m	1m 23s	7.8	NN_04_06_2022__16_11_28
4050	20.9	3.9	41.9	75.6	1h 33m	1m 57s	4.6	NN_04_06_2022__16_13_13
6075	17.0	3.2	35.5	69.7	2h 41m	3m 53s	2.5	NN_04_06_2022__16_15_20
9113	15.6	3.0	31.4	67.3	3h 15m	4m 55s	0.8	NN_04_06_2022__16_18_43
13666	12.4	2.4	24.8	60.3	5h 34m	6m 46s	0.5	NN_04_06_2022__16_24_40

Figura 3.13: File di log relativo al loop RETRAIN

3.2.2 Loop ZOOM

Il secondo ciclo viene effettuato tra il loop RETRAIN e lo step 3 (figura 3.14). Considerando che la complessità dell'addestramento della rete neurale dipende dalla quantità e dalla variabilità dei dati in input, tramite questo loop ad ogni iterazione viene ridotto il dominio all'interno del quale vengono selezionati randomicamente i valori dei raggi. In questo modo, lavorando con dataset caratterizzati da range di variabilità ristretti, è possibile ottenere delle reti sempre più affidabili. Il ciclo si articola nelle seguenti fasi:

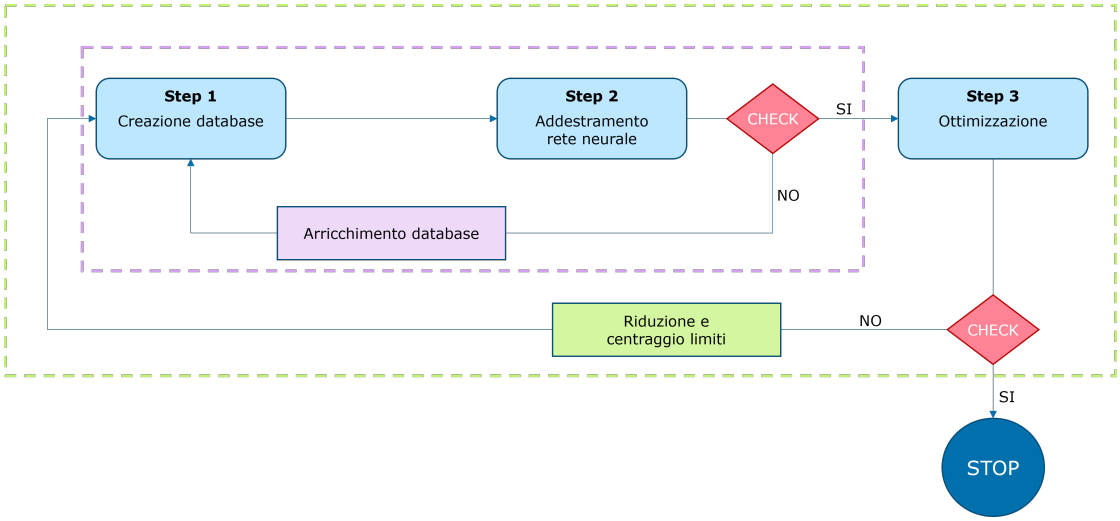

Figura 3.14: Loop ZOOM [[step1-step2]-step3]

1. Al termine del loop RETRAIN, si procede con l'utilizzo dell'Algoritmo Genetico per la ricerca dell'ottimo;

2. Salvataggio delle informazioni utili su un file di log (numero iterazione, numero di valutazioni della soluzione candidata durante l'ottimizzazione, durata dell'ottimizzazione, valore della massa, configurazioni dei raggi che minimizza la massa) (figura 3.15);

3. Viene effettuato un controllo sulla differenza tra i valori della funzione obiettivo corrispondenti alle ultime due iterazioni;

4. Impostazione del nuovo dominio per la generazione dei modelli. Il range viene centrato rispetto al risultato fornito dall'ottimizzatore all'iterazione precedente e viene ridotto di una certa percentuale, per ogni variabile di progetto. Viene permesso al nuovo dominio di oltrepassare i limiti imposti dall'ottimizzazione, per fare in modo che la rete possa indagare al meglio i valori nell'intorno della soluzione, considerando che il valore ottimo di alcune variabili potrebbe corrispondere al limite inferiore o superiore. Si precisa che il dominio di ottimizzazione, in cui opera l'algoritmo genetico, rimane sempre fisso;

5. Il processo riparte dal loop RETRAIN per una nuova iterazione e si torna al punto 1, fin quando due ottimizzazioni successive non restituiscono soluzioni che differiscono di una quantità inferiore rispetto ad un valore target.

Per permettere il corretto svolgimento del ciclo è necessario impostare alcuni parametri: per quanto riguarda la modifica del range da cui ricavare i dati in input, bisogna settare la percentuale di riduzione e una dimensione minima del dominio, per evitare che si restringa troppo; per quanto riguarda invece il controllo dei risultati, va definito il Δ limite tra Objective Function relative a due iterazioni successive, al di sotto del quale termina il processo.

iterazione	n° chiamate	tempo	massa[kg]	raggi[mm]
1	30203	19m 40s	55.69	[38.3, 12.3, 31.3, 25.1, 10.2, 14.5, 11.5, 31.4, 26.9, 12.5]
2	31253	19m 56s	52.0	[39.8, 10.1, 28.7, 24.5, 10.6, 10.7, 11.0, 27.9, 29.1, 10.4]
3	26761	17m 5s	50.36	[38.2, 11.1, 29.9, 24.3, 10.1, 10.3, 10.9, 28.4, 26.9, 10.7]
4	36803	23m 56s	48.77	[38.5, 10.6, 27.2, 22.8, 10.2, 10.2, 11.2, 28.8, 26.9, 10.3]
5	94703	1h 0m	47.88	[38.2, 10.0, 28.8, 23.2, 10.0, 10.0, 10.2, 27.2, 26.9, 10.0]
6	156192	1h 39m	47.92	[38.6, 10.0, 27.8, 22.9, 10.0, 10.0, 11.2, 27.4, 26.8, 10.0]

Figura 3.15: File di log relativo al loop ZOOM

Avendo effettuato più iterazioni per giungere all'ottimo, non si ha più un'unica rete neurale addestrata da interrogare durante il processo di ottimizzazione; allo stesso modo si hanno anche diversi scaler, la cui metodologia di normalizzazione è associata alle dimensioni e ai limiti inferiore e superiore del range degli input. Per questo motivo, il primo passaggio dello step 3 consiste nel caricare tante reti e tanti *scaler* pari al numero di iterazioni che sono state eseguite. Successivamente, all'interno dell'algoritmo genetico è necessario definire un controllo in modo da assegnare ad ogni soluzione candidata una funzione di normalizzazione ed una rete che siano adatte al dominio di cui fa parte (figura 3.16). Questo passaggio è fondamentale perché una rete addestrata con un dataset relativo ad un piccolo range di informazioni si comporta male al di fuori quel dominio, restituendo previsione errate.

Il loop ZOOM dà quindi la possibilità all'algoritmo di ottimizzazione di consultare reti neurali sempre più precise man mano che si avvicina all'ottimo.

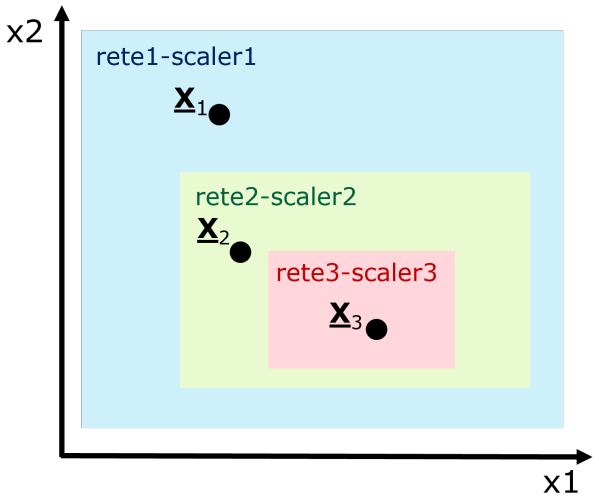

Figura 3.16: Schema di esempio per l'assegnazione dello scaler e della rete neurale per un dominio 2D

Prima di eseguire il run del codice, c'è la possibilità di scegliere se iniziare un nuovo processo di ottimizzazione o di proseguirne uno già iniziato. Infatti, se per qualsiasi motivo si volesse interrompere lo svolgimento, tramite la lettura dei vari file di log, INNO è in grado di riprenderlo successivamente, impostando in modo corretto numero dell'iterazione e dominio delle 10 variabili.

3.2.3 Analisi dei risultati

Grazie alla presenza dei cicli appena illustrati, il codice INNO permette di raggiungere risultati soddisfacenti ai fini dell'ottimizzazione.

Il primo beneficio è relativo al miglioramento dell'affidabilità delle reti neurali (figura 3.17), ottenuto grazie alla riduzione del dominio del dataset. Si può notare infatti che, tramite iterazioni successive, si ottiene una diminuzione considerevole dell'errore medio. In precedenza, utilizzando l'infrastruttura preliminare, la valutazione del massimo stress assiale veniva effettuata con una rete affetta da un errore del 2,5%; INNO consente di valutare l'output commettendo errori molto vicini allo 0%.

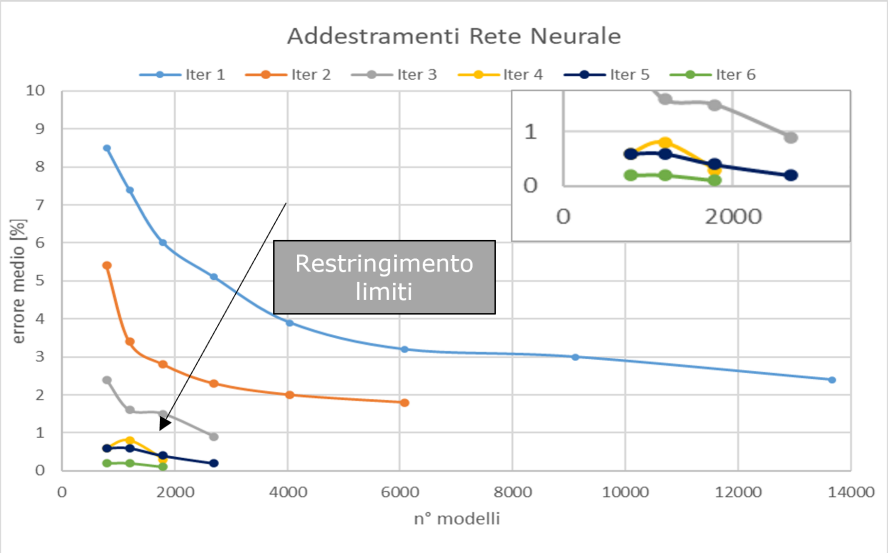

Figura 3.17: Analisi degli addestramenti delle reti neurali - Versione B

Dal grafico si può notare inoltre, che per l'addestramento di reti che devono lavorare su domini ridotti, è richiesto un numero inferiore di modelli. Questo è dovuto alla presenza di un campo ristretto di variabili in input e quindi la quantità di combinazioni randomiche possibili si abbassa drasticamente.

Il miglioramento della qualità degli addestramenti delle reti, comporta l'ottenimento di una soluzione al problema di ottimizzazione molto vicina al target di confronto, ossia il risultato fornito da Optistruct. Dalla tabella 3.8 è possibile notare che sulla maggior parte delle variabili di progetto viene commesso un errore inferiore all'1% e, più precisamente, 5 di queste hanno lo stesso valore proposto dal software. Dall'esito molto positivo di questi risultati, ne consegue che anche la massa dell'intera struttura, funzione obiettivo da minimizzare, raggiunge un valore concorde con quello di Optistruct.

		Optistruct	INNO (1 NN)	errore [%]
variabili [mm]	x1	38,6	38,6	0,0
	x2	10,0	10,0	0,0
	x3	27,6	27,8	0,7
	x4	23,0	22,9	-0,4
	x5	10,0	10,0	0,0
	x6	10,0	10,0	0,0
	x7	11,0	11,2	1,8
	x8	27,1	27,4	1,1
	x9	26,9	26,8	-0,4
	x10	10,0	10,0	0,0
objective function [kg]	massa	47,66	47,92	0,55

err < 1%	1% ≤ err < 3%	err ≥ 3%

Tabella 3.8: Confronto tra risultati di Optistruct e quelli di INNO

3.3 Versione C: INNO multi-rete

Il massimo stress assiale della struttura complessiva può riferirsi indistintamente ad una delle 10 barre e non sempre alla stessa. Considerando che, durante l'addestramento, la matrice Y fornita contiene semplicemente un valore, senza alcun riferimento al numero della barra, la previsione dell'output da parte della rete neurale risulta difficoltosa perché la relazione con gli input è molto complessa. Gli errori dovuti a questa impostazione del metodo hanno influenza sull'esito dell'ottimizzazione, la cui soluzione risulta leggermente diversa da quella ottenuta con Optistruct.

Per ovviare a questa problematica, si è scelto di introdurre nella metodologia l'utilizzo di più reti neurali, una per ogni variabile, mantenendo pressoché inalterata la struttura del processo iterativo, costituito dai loop RETRAIN e ZOOM (figura 3.18). Nei paragrafi che seguono verranno evidenziate le modifiche introdotte rispetto al processo caratterizzato da un'unica rete.

3.3.1 Loop RETRAIN

In prima battuta, è necessario che durante la creazione del database vengano letti i massimi stress assiali relativi a tutte le barre. Così facendo, al termine dello step 1, con lo stesso numero di modelli che venivano utilizzati per la versione precedente di INNO, si riescono ad ottenere 10 matrici Y. Si procede quindi con l'addestramento della prima rete neurale, impostando inizialmente il numero di modelli con cui si vuole iniziare; se necessario viene incrementato il database secondo le regole del loop RETRAIN spiegato in precedenza. Nel momento in cui è richiesta la

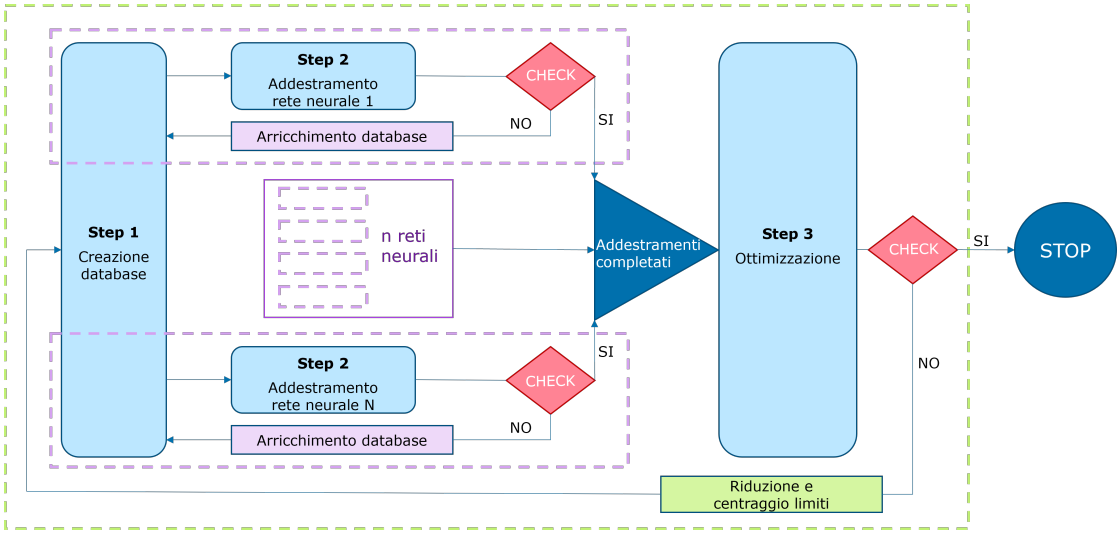

Figura 3.18: Struttura di INNO multi-rete

realizzazione e di conseguenza la simulazione di ulteriori modelli, vengono arricchite tutte le matrici Y e non solo quella che si riferisce all'addestramento in questione. In questo modo, se le reti successive dovessero necessitare di una quantità di dati maggiore rispetto a quella definita inizialmente, possono sfruttare gli incrementi già effettuati in precedenza, ottenendo un risparmio in termini di tempo.

Una volta soddisfatta la condizione del loop RETRAIN relativo alla prima rete neurale, si prosegue con gli altri addestramenti. Realizzati i modelli matematico-informatici per prevedere il massimo stress assiale di ciascun segmento della struttura, si può procedere con l'impostazione dell'ottimizzazione.

3.3.2 Loop ZOOM

Per quanto riguarda la struttura del loop ZOOM, non sono state apportate modifiche; infatti la presenza di più reti non influenza in alcun modo la verifica effettuata sul risultato dell'ottimizzazione.

L'unico cambiamento viene realizzato sul calcolo degli stress e sulla verifica dei vincoli di progetto. Potendo prevedere il massimo stress assiale di ciascuna barra, per ogni soluzione candidata viene richiesto l'output di tutte e 10 le reti, da cui si ricava il massimo, che deve rispettare il limite strutturale imposto.

Per assicurarsi che il risultato fornito dall'algoritmo di ottimizzazione sia valido, è stata implementata una funzione al fine di monitorare il corretto utilizzo delle reti neurali. Per far sì che la verifica vada a buon fine è necessario che tutti gli elementi dell'array soluzione siano contenuti all'interno del dominio con cui è stata addestrata la rete neurale relativa all'ultimo step iterativo. Se ciò non dovesse accadere, verrebbe controllata l'appartenenza al dominio caratteristico della penultima iterazione; una volta verificata questa condizione, la soluzione è considerata valida solo se la funzione obiettivo è inferiore rispetto a quella fornita dall'algoritmo all'iterazione precedente.

É presente una porzione del codice per poter comprendere meglio il funzionamento del processo per la verifica della soluzione.

```
if iterazione == 1:
    verifica = True

else:
    count_penultimo = 0
    penultimo_lb = lista_lb[-2]
    penultimo_ub = lista_ub[-2]
    for k in range(n_variabili):
        if penultimo_lb[k]<=x[k]<=penultimo_ub[k]:
            count_penultimo += 1

    count_ultimo = 0
    ultimo_lb = lista_lb[-1]
    ultimo_ub = lista_ub[-1]
    for k in range(n_variabili):
        if ultimo_lb[k]<=x[k]<=ultimo_ub[k]:
            count_ultimo += 1


    if count_ultimo == n_variabili:
        verifica = True
    elif count_ultimo < n_variabili \
        and count_penultimo == n_variabili and fun < fun_1:
        verifica = True
    else:
        verifica = False
```

3.3.3 Analisi dei risultati

Come fatto per la versione B del codice, che presentava una sola rete neurale per il calcolo del massimo stress assiale, anche per la versione C (INNO multi-rete) vengono analizzati sia i risultati relativi agli addestramenti delle reti, sia quelli che si riferiscono al risultato finale dell'ottimizzazione.

Addestrando delle reti neurali in cui la relazione tra variabili in input e output è meno complessa rispetto al caso con un'unica rete, poiché ognuna di esse controlla il comportamento di una specifica barra, è sufficiente un database di dimensioni più piccole. Infatti, dai grafici in figure 3.19 e 3.20 è possibile notare che si ottiene un'elevata affidabilità della rete con un numero di modelli inferiore rispetto alla versione B del codice, permettendo di ridurre il tempo dedicato all'utilizzo del software. Per di più si raggiunge un errore medio vicino allo 0% dopo sole 3 iterazioni.

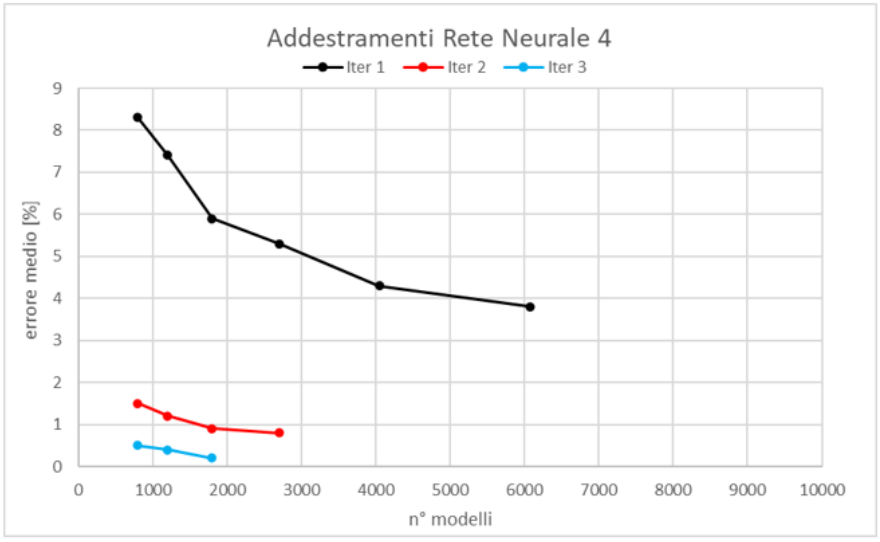

Figura 3.19: Addestramenti della rete neurale 4 - Versione C

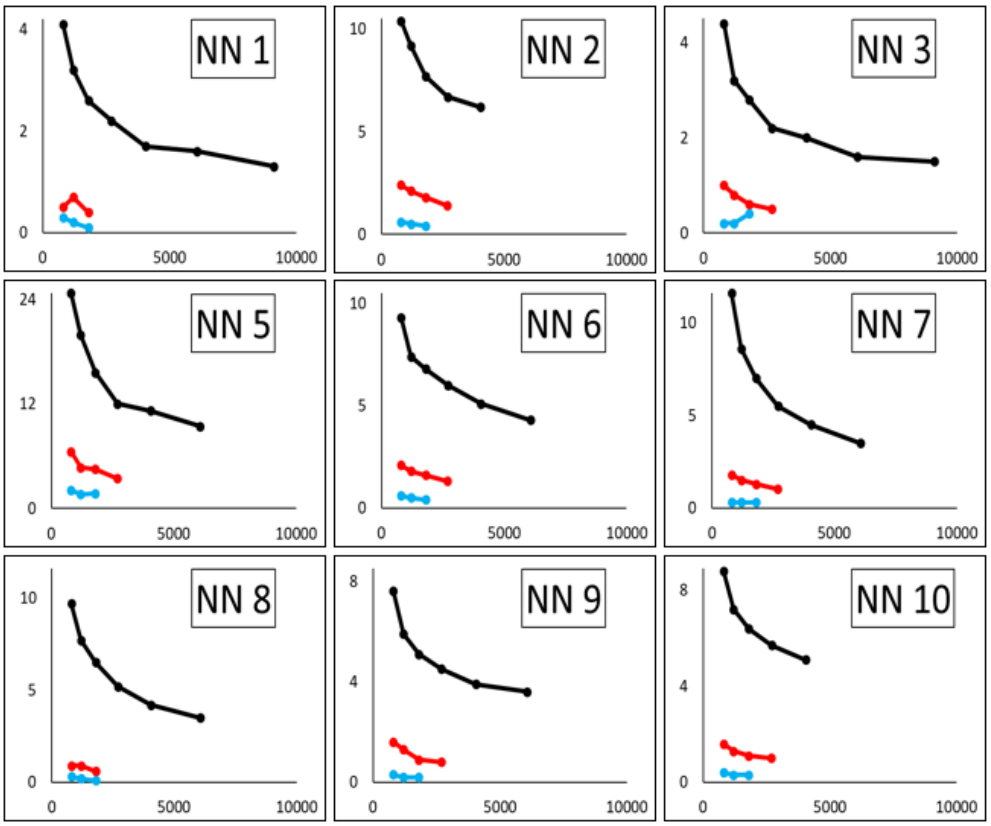

Figura 3.20: Addestramenti delle reti neurali di INNO multi-rete

In virtù dei risultati soddisfacenti degli addestramenti, la metodologia che contempla l'utilizzo di 10 reti neurali conduce ad un valore della massa della struttura (Objective Function) uguale a quello ottenuto attraverso il solutore di Altair (tabella 3.9). Nonostante si raggiunga l'ottimo grazie ad una configurazione degli spessori delle barre differente rispetto a quella proposta da Optistruct, la soluzione viene considerata valida. L'obiettivo è infatti quello di minimizzare la massa, indipendentemente dalla dimensione delle sezioni.

In aggiunta alla alta qualità dei risultati, le reti neurali multiple consentono di risolvere problemi in cui vengono utilizzati più materiali differenti. Calcolando lo stress a cui è sottoposto ogni elemento di un sistema, è possibile imporre i vincoli strutturali in relazione alle proprietà meccaniche caratteristiche dei singoli componenti. Tale impostazione del codice permette di affrontare problemi in cui, oltre alla ricerca dei parametri geometrici ottimali, si aggiunge la variabilità dei materiali, realizzando un'ottimizzazione più completa.

		Optistruct	INNO (10 NNs)	errore [%]
variabili [mm]	x1	38,6	38,6	0,0
	x2	10,0	10,1	1,0
	x3	27,6	27,7	0,4
	x4	23,0	22,8	-0,9
	x5	10,0	10,1	1,0
	x6	10,0	10,0	0,0
	x7	11,0	10,3	-6,4
	x8	27,1	27,3	0,7
	x9	26,9	26,7	-0,7
	x10	10,0	10,9	9,0
objective function [kg]	massa	47,66	47,66	0,00

err < 1%	1% ≤ err < 3%	err ≥ 3%

Tabella 3.9: Confronto tra risultati di Optistruct e quelli di INNO multi-rete

3.4 Confronto dei tempi di processo e dei risultati

Dopo aver spiegato il funzionamento delle diverse versioni del codice e aver mostrato i risultati a cui conducono, è interessante effettuare un confronto per la valutazione dei tempi di processo (tabella 3.10). Per quanto riguarda la metodologia illustrata in questo elaborato, vengono valutati i codici INNO e INNO multi-rete; questi sono messi a paragone con un processo nel quale l'algoritmo genetico si serve degli output forniti da Altair Optistruct. Un confronto con le tempistiche dell'ottimizzatore interno al software non avrebbe rilevanza, in quanto non permette di affrontare problemi che invece potrebbero essere risolti con la metodologia sviluppata. Infatti,

	GA/FEA	INNO	INNO multi-rete
Iterazioni (step 1 - step 2 - step 3)	1	6	3
Numero di modelli		32115	13611
Tempo creazione Database		40h 6m	15h 35m
Tempo Addestramenti		39m 5s	1h 45m
Valutazioni della soluzione candidata	54600	375915	225298
Tempo Ottimizzazione	77h	3h 59m	24h 12m
Tempo di Processo totale	77h	44h 44m	41h 32m
Δ Tempo di Processo totale [%]		-41,9 %	-46,1 %
Δ Tempo di utilizzo della Licenza del Solutore [%]		-47,9 %	-79,8 %

Tabella 3.10: Confronto tra i tempi di processo relativi alle diverse metodologie

è stato scelto un caso di studio test risolvibile con un applicativo commerciale solamente per avere una soluzione con cui validare il codice.

Si precisa che l'impiego delle reti neurali in sostituzione al solutore richiede la presenza di due step preparativi, prima di utilizzare l'algoritmo di ottimizzazione: la creazione del database e l'addestramento. Bisogna quindi considerare il tempo per lo svolgimento di alcuni processi, che invece non vengono contemplati quando ci si serve del software. Inoltre, per poter raggiungere un'elevata affidabilità delle reti, ed ottenere quindi una soluzione attendibile, sono necessarie diverse iterazioni.

Di seguito è presente un confronto tra le 3 metodologie, relativamente ai diversi step del processo:

1. L'utilizzo di un'unica rete neurale rende la predizione dell'output più complessa, riferendosi all'intera struttura e non alle singole barre. Per poter raggiungere prestazioni soddisfacenti, sono richiesti molti più modelli rispetto ad INNO multi-rete ed un maggior numero di ripetizioni dei due loop caratteristici della metodologia. Dovendo realizzare un database più vasto, il tempo di utilizzo della licenza di Altair Optistruct è pari a 40 ore e 6 minuti, a fronte di 15 ore e 35 minuti, tempo utilizzato invece per le analisi dei modelli necessari per gli addestramenti delle 10 reti. Il risparmio ottenuto nella prima fase del processo non è assolutamente paragonabile con la maggior quantità di tempo da dedicare agli addestramenti, considerando che impiegano pochi minuti l'uno;

2. per quanto riguarda lo step 3, ossia l'ottimizzazione, la risoluzione tramite l'accoppiamento algoritmo/software richiede 77 ore, per via del fatto che per valutare la soluzione candidata deve essere richiamato il solutore, che effettua di volta in volta l'analisi. Grazie all'utilizzo delle reti neurali la valutazione della possibile soluzione viene compiuta in qualche decimo di secondo, riducendo drasticamente la durata di questa fase. In questo modo, sfruttando un modello matematico-informatico per la predizione dell'output, la durata

dell'ottimizzazione non dipende dalla complessità dell'analisi, diversamente dal caso in cui l'algoritmo genetico si serve dei risultati forniti dal software. La metodologia, quindi, risulta ancor più vantaggiosa quando si lavora con modelli complessi, che richiedono maggiori tempi di calcolo. Riguardo le due versioni di INNO, quella che sfrutta più reti impiega più tempo, dovendo effettuare ogni volta la previsione di 10 output e non del massimo stress assoluto;

3. nel calcolo del tempo totale, le differenze di tempistiche tra INNO e INNO multi-rete, relative agli step 1 e 3, si bilanciano, conducendo ad una durata dell'intero processo molto simile. In entrambe i casi si riesce a risparmiare più del 40% del tempo rispetto all'altra strategia risolutiva;

4. servendosi del modello surrogato realizzato con le reti neurali, i tempi relativi all'utilizzo del software risultano ridotti, perché necessario solamente nella prima fase di ogni iterazione; l'altro metodo invece richiede di dedicare la licenza per tutto il tempo del processo, rappresentando un vincolo. In termini percentuali, sfruttando diverse reti neurali in parallelo si riesce a ridurre il tempo di utilizzo del solutore dell'80%.

Dopo aver dimostrato che il codice realizzato permette di ridurre i tempi di processo, è opportuno verificare che i risultati restituiti siano coerenti con quelli delle simulazioni. Come mostrato già in precedenza, in entrambe i casi il valore della funzione obiettivo al termine dell'ottimizzazione è molto vicino, o uguale, a quello atteso. L'affidabilità delle reti neurali viene valutata tramite il coefficiente di sicurezza, calcolato come il rapporto tra la σ di snervamento e lo stress massimo associato alla configurazione delle barre proposta, calcolato effettuando un'analisi con il software. Affidandosi ad una sola rete per la previsione dell'output, non si ottiene la stessa precisione che viene raggiunta con INNO multi-rete nel calcolo dello stress massimo. La soluzione proposta dall'ultima versione del codice conduce ad un errore sulla massa dello 0% e ad un coefficiente di sicurezza pari ad 1, nonostante alcuni valori dei raggi siano differenti rispetto a quelli suggeriti da Optistruct.

Le considerazioni appena presentate evidenziano il grande vantaggio di poter risolvere problemi di ottimizzazione non gestibili con applicativi commerciali, riducendo i tempi di calcolo e di uso licenza rispetto ad un possibile accoppiamento tra algoritmo genetico e software CAE. Questo beneficio

In base ai risultati ottenuti in termini di tempi e precisione della soluzione, il codice INNO multi-rete (tabella 3.11) è stato ritenuto il più valido e verrà quindi utilizzato per la risoluzione di problemi di ottimizzazione d'interesse industriale. Intanto nel capitolo successivo verranno illustrate due applicazioni a situazioni non affrontabili con un solutore commerciale (Altair Optistruct).

	file .py	n° righe	n° righe per step
main	MAIN	237	397
	funzioni	160	
step1	step1_main	115	217
	step1_funzioni	102	
step 2	step2_main	150	285
	step2_funzioni	135	
step 3	step3_main	150	210
	step3_funzioni	60	
TOTALE		1109	

Tabella 3.11: Numero di righe del codice

Capitolo 4

Applicazioni del codice INNO multi-rete

Dopo aver consolidato e validato la metodologia, INNO multi-rete è stato utilizzato per ricercare la soluzione a due problemi di ottimizzazione, non risolvibili tramite software commerciali. Il primo problema richiede di lavorare con variabili di progetto discrete, mentre il secondo è caratterizzato dall'interazione tra due discipline.

4.1 Problema di ottimizzazione con variabili di progetto discrete

In ambito industriale, non è sempre possibile produrre componenti di dimensioni variabili in modo continuo. Spesso capita che sia possibile realizzare un prodotto solamente in determinate misure, rispettando intervalli ben precisi. In questi casi è necessario basare l'ottimizzazione su variabili discrete, perché la considerazione di un dominio continuo e la successiva approssimazione della soluzione potrebbe condurre ad un risultato che non rispetta i vincoli progettuali o lontano dall'ottimo.

Il problema in questione prende spunto dal caso di studio test, utilizzato per lo sviluppo del codice. Si considera, infatti, lo stesso modello descritto in precedenza, ossia una struttura reticolare composta da 10 barre a sezione circolare, incastrata ad un'estremità e sollecitata all'estremo opposto (figura 2.1). L'unica differenza è dovuta all'ipotesi secondo cui i vari segmenti possano essere prodotti con raggi variabili da 10 mm a 60 mm, con passo di 10 mm. L'impostazione del problema di ottimizzazione, per quanto riguarda funzione obiettivo e vincoli di progetto rimane invariata (tabella 4.1).

Nella risoluzione con Altair Optistruct si incontrano problemi legati al dominio discreto delle variabili di progetto. Questo comporta la discontinuità della funzione obiettivo, che rende difficoltosa un'ottimizzazione che si serve di un algoritmo gradient-based, come nel caso del solutore commerciale a disposizione. É probabile che durante il processo l'algoritmo conduca ad un minimo locale, dove si stabilizza. Essendo quella da minimizzare una funzione in 10 variabili, la difficoltà risiede

VARIABILI	$\{x_i\} \quad with\ i = 1, \dots, 10$	raggi delle barre
BOUNDS	$10\ mm \leq x_i \leq 60\ mm$	i raggi delle sezioni circolari possono variare da 10 mm a 60 mm, in modo discreto ad intervalli di 10 mm
OBJECTIVE FUNCTION	$obj(x_i) = m_{TOT} = \left(\sum_{i}^{10} (\pi x_i^2) l_i\right) \rho$	la funzione da minimizzare corrisponde alla massa complessiva della struttura
CONSTRAINTS	$\lvert\sigma_{max}\rvert < \sigma_y$	la tensione assiale massima deve essere inferiore alla tensione di snervamento

Tabella 4.1: Sintesi del problema di ottimizzazione con variabili discrete

nella scelta della configurazione iniziale delle variabili, ovvero l'array x_0, in modo da poter raggiungere il minimo globale.

		valore iniziale dei raggi [mm]										soluzione [mm]										massa [kg]	violazione vincoli
		$x1_0$	$x2_0$	$x3_0$	$x4_0$	$x5_0$	$x6_0$	$x7_0$	$x8_0$	$x9_0$	$x10_0$	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10		
ottimizzazioni	1	10	10	10	10	10	10	10	10	10	10	40	20	40	20	10	20	20	30	30	20	71,41	SI
	2	20	20	20	20	20	20	20	20	20	20	40	20	40	20	10	20	20	30	30	20	71,41	SI
	3	60	60	60	60	60	60	60	60	60	60	40	30	40	20	10	30	20	20	20	20	67,8	SI
	4	40	20	30	30	10	10	10	30	30	10	40	10	30	30	10	10	10	30	30	10	57,02	NO

Tabella 4.2: Soluzioni fornite da Optistruct in funzione della x_0

In tabella 4.2 sono riassunte 4 ottimizzazioni realizzate in Altair Optistruct, caratterizzate da differenti valori iniziali delle variabili. Si nota che, oltre a fornire risultati differenti, sia in termini di raggi che di massa, nella maggior parte dei casi vengono violati i vincoli di progetto, in quanto lo stress assiale massimo eccede il limite di snervamento imposto. L'unica x_0 che permette di raggiungere la massa minima rispettando il limite strutturale è quella relativa all'ultima riga della tabella. Questo array di valori iniziali è stato definito successivamente alla risoluzione del problema tramite la versione A del codice, che sfrutta l'accoppiamento tra l'algoritmo genetico e la rete neurale. Essendo un problema complesso, in cui la relazione tra input e output non è di semplice interpretazione, è improbabile che l'utente possa proporre una x_0 come quella presente nell'ultima riga.

La tabella appena analizzata dimostra l'importanza della scelta dei valori di partenza, risolvendo un problema di ottimizzazione in Altair Optistruct. La complessità di tale selezione è proporzionale al numero di variabili e alla grandezza del dominio.

Per far fronte a queste difficoltà, si è ricorso all'utilizzo di INNO multi-rete per ricercare la soluzione al problema proposto. Sfruttando un algoritmo genetico, non basato sul calcolo gradiente ma su tecniche stocastiche, il rischio di stabilizzarsi in

un minimo locale si riduce di molto; inoltre, la popolazione iniziale viene impostata secondo metodi randomici o statistici, limitando la responsabilità dell'utente. Dopo sole 2 iterazioni, il codice fornisce una soluzione a cui è associata la massa minima e che rispetta i vincoli strutturali (tabella 4.3).

		Optistruct	INNO (10 NNs)	errore [%]
variabili [mm]	x1	40	40	0,0
	x2	20	20	0,0
	x3	30	30	0,0
	x4	30	30	0,0
	x5	10	10	0,0
	x6	10	10	0,0
	x7	10	10	0,0
	x8	30	30	0,0
	x9	30	30	0,0
	x10	10	10	0,0
objective function [kg]	massa	57,02	57,02	0,00

err < 1%	1% ≤ err < 3%	err ≥ 3%

Tabella 4.3: Soluzione dell'ottimizzazione con variabili discrete

La configurazione di raggi che minimizza la massa viene raggiunta già dopo la prima iterazione, ma la seconda è necessaria per verificare se tramite i loop di convergenza si ottiene una variazione del risultato.

4.2 Problema di ottimizzazione multidisciplinare

Un altro problema di interesse industriale, risolto tramite INNO multi-rete, riguarda l'ottimizzazione parametrica della portiera di un furgone. Il caso di carico analizzato consiste nel fornire un'accelerazione negativa in direzione z pari a 3g e un'accelerazione negativa in direzione y pari a 1g (figura 4.1). Si è assunto che la portiera sia aperta e incastrata all'estremità in corrispondenza di 4 punti. In questo caso la funzione da minimizzare è relativa al costo di produzione, che deve essere ottimizzato facendo variare gli spessori dei vari componenti entro determinati limiti. L'inclusione dell'analisi economica all'interno di una verifica strutturale rende il problema multidisciplinare, non risolvibile tramite software commerciali. Si precisa che nel calcolo non vengono considerati i costi fissi e quelli relativi all'assemblaggio, ma solamente ciò che è legato alla produzione dei singoli componenti. Si fa riferimento quindi al costo della materia prima, del processo di produzione e delle lavorazioni necessarie. L'obiettivo è, infatti, dimostrare le potenzialità della metodologia; la definizione esatta di tutti i contributi che concorrono al calcolo

del costo di produzione non è d'interesse. La tabella 4.4 riassume il problema di ottimizzazione in questione.

VARIABILI	$\{x_i\} \quad with\, i = 1, \dots, 9$	spessori dei componenti
BOUNDS	$\boldsymbol{x}_{inf} = [0.5, 0.6, 0.9, 0.6, 1.1, 0.9, 1.1, 0.5, 0.5]$ $\boldsymbol{x}_{sup} = [1.1, 1.0, 1.6, 1.2, 1.8, 1.6, 1.8, 1.1, 1.0]$	gli spessori possono variare all'interno di determinati limiti, in base alla funzione del componente. I valori sono in mm
OBJECTIVE FUNCTION	$obj(x_i) = costo = \sum_{i=1}^{9} fun(x_i)$	la funzione da minimizzare corrisponde al costo di produzione
CONSTRAINTS	$(\sigma_{max}^{VM})_i < (\sigma_y)_i \quad with\, i = 1, \dots, 9$	la massima tensione secondo VonMises di ogni componente deve essere inferiore al limite di snervamento del materiale

Tabella 4.4: Sintesi del problema di ottimizzazione multidisciplinare

La portiera, oggetto del problema, è costituita da 10 componenti: 9 di questi devono essere ottimizzati, mentre un pannello realizzato in vetroresina è caratterizzato da una serie di spessori variabili, che non possono essere modificati (figura 4.1,tabella 4.5). Non essendo oggetto dell'ottimizzazione, tale componente non viene considerato nel calcolo del costo di produzione.

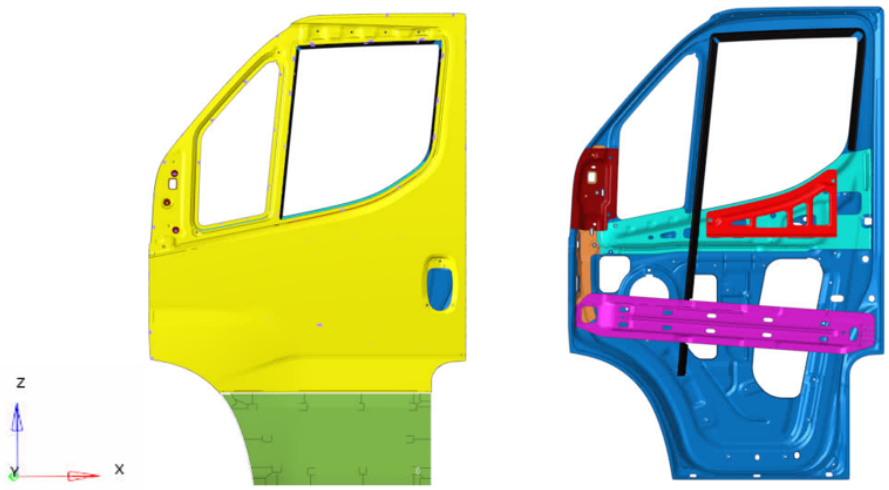

Figura 4.1: Componenti portiera

Nella fabbricazione della portiera vengono impiegati tre materiali differenti, con determinate caratteristiche meccaniche, in base alle quali bisogna effettuare la verifica strutturale (tabella 4.6). In questa prima fase i materiali sono fissati; in un secondo momento verranno inseriti come ulteriori variabili di progetto. Una funzione obiettivo relativa ai costi di produzione sarà utile per comprendere se possa essere vantaggioso utilizzare materiali con migliori caratteristiche meccaniche, ma allo stesso tempo più costosi, o viceversa.

		Spessore		
		minimo [mm]	nominale [mm]	massimo [mm]
	Cornice	0,5	0,8	1,1
	Cover esterna	0,6	0,7	1,0
	Serratura	0,9	1,2	1,6
	Anima sotto finestrino	0,6	0,8	1,2
	Lamiera	1,1	1,5	1,8
	Sede maniglia	0,9	1,2	1,6
	Anima lato finestrino	1,1	1,5	1,8
	Lamiera triangolare	0,5	0,7	1,1
	Cover interna	0,5	0,8	1,0
	Pannello inferiore	non è una variabile di progetto		

Tabella 4.5: Componenti portiera

ID materiale	Nome materiale	Tipologia materiale	E [GPa]	v	ρ [kg/m^3]	σ_y [MPa]	costo [€/kg]
1	S355MC	acciaio	207,5	0,29	7850	355	13,26
2	DC 04	acciaio basso carbonio	210	0,29	7860	210	12,86
3	smc	vetroresina	11	0,37	1800	73	n.d.

Tabella 4.6: Caratteristiche dei materiali utilizzati

Prima di procedere con la risoluzione del problema, si è scelto di validare nuovamente il codice impostando come funzione obiettivo una quantità che potesse essere monitorata anche da Altair Optistruct, ovvero la massa dell'intero sistema. É stata sfruttata di fatto la versione C del codice illustrata in precedenza, effettuando le modifiche del caso, relative ai limiti inferiore e superiore, ai limiti strutturali, al numero di variabili e alla funzione per il calcolo della massa, che in questo caso deve fare riferimento alle superfici di ogni componente. Inoltre, dovendo confrontare la tensione massima con il limite di snervamento caratteristico dei materiali utilizzati, non è possibile avvalersi di una rete neurale che fornisce un unico output, ma è necessario addestrarne una per ogni elemento della portiera. Nonostante il pannello in vetroresina non sia oggetto dell'ottimizzazione, è comunque richiesta una rete dedicata alla previsione della relativa tensione secondo Von Mises, per verificare il

superamento o meno del limite strutturale.

Essendo un problema in cui la relazione tra le variabili di progetto in input e gli stress in output è meno complessa rispetto al problema delle barre, è stato sufficiente un numero inferiore di modelli. Già alla prima iterazione, con soli 2500 modelli a disposizione, tutte le reti risultano affette da un errore molto vicino allo 0%. Questo comporta vantaggi nella risoluzione dell'ottimizzazione, ottenendo, dopo 4 iterazioni, un valore della funzione obiettivo in linea con il target fornito da Altair Optistruct. Osservando la tabella 4.7 si nota che viene commesso un errore importante solamente su un componente. Si tratta della sede della maniglia, che ha un peso trascurabile nel calcolo della massa dell'intera portiera rispetto a componenti più grandi, come ad esempio le cover e la cornice. Difatti la funzione obiettivo è affetta da un errore accettabile, pari allo 0,88% (tabella 4.8). La causa di questa differenza tra le due soluzioni è da attribuire molto probabilmente alla natura stocastica dell'algoritmo di ottimizzazione, in quanto utilizzando la soluzione di Optistruct come input delle reti neurali, tutti e 10 le tensioni di Von Mises massime risultano uguali a quelle calcolate tramite il software. Per ottenere soluzioni ancor più precise, verrà effettuata un'indagine sui parametri caratteristici dell'algoritmo e su altre potenziali metodologie gradient-free.

	Spessore		
	Optistruct [mm]	INNO [mm]	Δ [%]
Cornice	0,50	0,52	4,00
Cover esterna	0,60	0,60	0,00
Serratura	1,60	1,57	-1,88
Anima sotto finestrino	0,60	0,61	1,67
Lamiera	1,10	1,11	0,91
Sede maniglia	0,90	0,98	8,89
Anima lato finestrino	1,80	1,79	-0,56
Lamiera triangolare	0,50	0,50	0,00
Cover interna	0,68	0,69	1,69
Pannello inferiore	non è una variabile di progetto		

err < 1%	1% ≤ err < 3%	err ≥ 3%

Tabella 4.7: Ottimizzazione della massa della portiera. Confronto tra risultati di Optistruct e quelli di INNO multi-rete

	Optistruct	INNO	Δ [%]
MASSA [kg]	18,17	18,33	0,88

Tabella 4.8: Confronto tra i valori di massa ottenuti con Optistruct e con INNO multi-rete

Una volta confermata l'affidabilità del codice, si è proceduto con l'impostazione del problema di ottimizzazione associato all'analisi economica. Per la definizione del costo di produzione di ogni componente, è stato realizzato un digital twin tramite il software di Siemens *Product Cost Management (PCM)*. Il tool permette di prevedere il costo relativo al processo produttivo, inserendo le seguenti informazioni: geometria del componente, numero di pezzi da realizzare, materiale, tecnologia produttiva, lavorazioni da eseguire. Per ogni componente sono state effettuate previsioni con spessori medi differenti all'interno del range di variabilità, in modo da ottenere una curva che rappresentasse l'andamento del costo in relazione dello spessore. Vengono quindi definite 9 funzioni che, inserite all'interno dell'algoritmo di ottimizzazione, contribuiscono al calcolo della funzione obiettivo, associata al costo di produzione dell'intera portiera (tabella 4.9).

	ID materiale	funzione di costo	costo nominale [€/pz]
Cornice	2	$c = -0.3372s^2 + 14.304s + 6.675$	17,89
Cover esterna	1	$c = 125.95s + 7.7814$	95,95
Serratura	1	$c = 0.0894s^2 + 6.184s + 5.1546$	12,70
Anima sotto finestrino	1	$c = 20.723s + 7.26$	23,84
Lamiera	1	$c = 0.127s^2 + 4.4619s + 4.8433$	11,56
Sede maniglia	2	$c = 0.1649s^2 + 2.2833s + 4.4674$	7,44
Anima lato finestrino	1	$c = 0.1631s^2 + 4.1945s + 6.039$	12,70
Lamiera triangolare	1	$c = -0.5552s^2 + 6.8682s + 4.5784$	9,11
Cover interna	1	$c = 0.0119s2 + 130.33s + 7.8243$	112,10
Pannello inferiore	3	n.d.	n.d.

Tabella 4.9: Funzioni di costo associate ai componenti della portiera, in relazione al materiale utilizzato

Dopo aver definito le curve di costo per tutti i componenti, è stata implementata la funzione obiettivo relativa al calcolo del costo di produzione. Per fare ciò, sono

stati inseriti i coefficienti delle funzioni di costo tra le variabili in input dello script relativo allo step 3. Per consentire all'algoritmo genetico di muoversi meglio all'interno del dominio, si è scelto di impostare una riduzione dei limiti del loop ZOOM meno aggressiva rispetto al problema precedente. In questo modo durante l'ottimizzazione viene interrogato un maggior numero di reti neurali, caratterizzate da un errore medio inferiore man mano che la soluzione candidata si avvicina al risultato finale. Dopo 6 iterazioni si giunge alla soluzione, mostrata in tabella 4.10. L'ottimizzazione ha avuto esito positivo in quanto, al termine del processo, lo spessore dei componenti sottoposti a sollecitazioni ridotte, ai quali corrispondono margini di sicurezza elevati, è uguale al limite inferiore, o molto vicino. Per i componenti sollecitati maggiormente, invece, viene proposto uno spessore che permetta di raggiungere un margine di sicurezza pari a 1. Fa eccezione l'anima posta al lato del finestrino, che ha un ruolo rilevante dal punto di vista strutturale.

	Spessore			MSy	costo [€/pz]
	minimo [mm]	massimo [mm]	ottimo [mm]		
Cornice	0,50	1,10	0,51	1,94	13,87
Cover esterna	0,60	1,00	0,60	2,93	83,38
Serratura	0,90	1,60	1,53	1,04	14,83
Anima sotto finestrino	0,60	1,20	0,60	6,23	19,71
Lamiera	1,10	1,80	1,23	2,32	10,35
Sede maniglia	0,90	1,60	0,94	6,77	7,89
Anima lato finestrino	1,10	1,80	1,80	2,48	14,12
Lamiera triangolare	0,50	1,10	0,50	16,14	7,87
Cover interna	0,50	1,00	0,68	1,01	96,49
Pannello inferiore	non è una variabile di progetto			36,50	n.d.
COSTO TOTALE					268,52 €

Tabella 4.10: Ottimizzazione del costo di produzione dei componenti della portiera

Per incrementare la complessità del problema e renderlo ancor più interessante dal punto di vista industriale, è stata considerata la variabilità dei materiali. Si è ipotizzato che tutti i componenti della portiera, ad eccezione del pannello in vetroresina, possano essere realizzati indipendentemente con il materiale 1 o con il materiale 2, illustrati in tabella 4.6. Osservando le caratteristiche meccaniche, si

nota che l'S355MC presenta un limite di snervamento maggiore, a discapito però di un costo al kilogrammo più alto. L'obiettivo dell'ottimizzazione è quindi quello di ricercare la combinazione ottima di spessori e materiali, al fine di ottenere un giusto compromesso tra costo e resistenza. A questo punto, le variabili di progetto raddoppiano perché si aggiungono gli ID dei materiali dei 9 componenti, che possono assumere il valore 1 o 2. É stato realizzato un file Excel che contiene i coefficienti delle funzioni di costo e le caratteristiche meccaniche relative ai materiali utilizzati. Per ogni soluzione candidata proposta dall'algoritmo genetico, vengono estrapolate tali informazioni in base all'ID del materiale, e calcolati di conseguenza costi di produzione e margini di sicurezza.

	Ottimizzazione Optistruct Materiale 1 Più performante, ma più costoso			Ottimizzazione Optistruct Materiale 2 Meno performante, ma più economico		
	soluzione Materiale 1 S355MC [mm]	costo di produzione [€/pz]	MS_y	soluzione Materiale 2 DC04 [mm]	costo di produzione [€/pz]	MS_y
Cornice	0,50	13,95	3,4	0,50	13,74	2,0
Cover esterna	0,60	83,35	2,9	0,60	81,17	1,7
Serratura	1,60	15,28	1,1	1,60	14,99	0,7
Anima sotto finestrino	0,60	19,69	6,1	0,60	19,34	4,9
Lamiera	1,10	9,91	1,5	1,23	10,55	1,0
Sede maniglia	0,90	7,12	11,4	0,90	7,05	7,0
Anima lato finestrino	1,80	14,12	2,5	1,80	13,88	1,7
Lamiera triangolare	0,50	7,87	15,2	0,50	7,79	8,4
Cover interna	0,68	96,19	1,0	1,00	134,40	0,7
Pannello inferiore	n.d.	n.d.	26,1	n.d.	n.d.	33,2

$MS_y > 1,5$	$1,5 \geq MS_y \geq 1$	$MS_y < 1$

Tabella 4.11: Ottimizzazioni preliminari effettuate con Optistruct

Per giustificare l'effettiva utilità di questa impostazione, vengono presentati i risultati relativi a due ottimizzazioni preliminari effettuate in Optistruct, settando per tutti i componenti oggetto di studio prima solo il materiale 1 e successivamente solo il 2 (tabella 4.11). Avendo utilizzato il solutore commerciale per la risoluzione del problema, è stato necessario impostare la massa come funzione obiettivo ed applicare in seguito le curve di costo alla soluzione per ricavare il costo di produzione totale. Impiegando il materiale 1 l'ottimizzazione ha esito positivo; per quanto riguarda il caso relativo al materiale 2, non si riesce a giungere ad una soluzione coerente con i vincoli di progetto. Infatti la serratura e la cover esterna, pur

caratterizzate dal massimo spessore possibile, sono sottoposte ad una tensione più elevata rispetto al limite di snervamento.

Procedendo con l'ottimizzazione tramite il codice INNO multi-rete, si ottiene una soluzione che contempla l'utilizzo del materiale più prestante per i componenti che nella seconda ottimizzazione effettuata con Optistruct presentano un margine di sicurezza inferiore o uguale a 1. Per tutti gli altri componenti conviene fare uso del materiale più economico per abbattere i costi di produzione (tabella 4.12). La soluzione di questo problema non era banale, per via del fatto che, con un caso di carico basato su accelerazioni, le sollecitazioni a cui è soggetta l'intera struttura sono fortemente dipendenti dalla densità dei materiali utilizzati. Grazie alla configurazione di spessori e materiali ottenuta con INNO multi-rete è possibile ridurre il costo totale, rispetto ai casi di studio risolti tramite Optistruct (tabella 4.13).

	Ottimizzazione INNO multi-rete			
	ID materiale	spessore [mm]	costo di produzione [€/pz]	MS_y
Cornice	2	0,51	13,88	1,9
Cover esterna	2	0,60	81,17	1,9
Serratura	1	1,59	15,21	1,1
Anima sotto finestrino	2	0,61	19,54	3,4
Lamiera	1	1,60	12,31	2,7
Sede maniglia	2	0,93	7,12	6,7
Anima lato finestrino	2	1,80	13,88	1,6
Lamiera triangolare	2	0,51	7,85	8,9
Cover interna	1	0,66	93,85	1,0
Pannello inferiore	3	n.d.	n.d.	30,4

$MS_y > 1,5$	$1,5 \geq MS_y \geq 1$	$MS_y < 1$

Tabella 4.12: Ottimizzazione del costo di produzione con materiali variabili, effettuata con INNO multi-rete

	Materiale 1 Optistruct	Materiale 2 Optistruct	Materiali variabili INNO multi-rete
COSTO TOTALE [€]	267,48	302,90	264,81
Rispetto dei vincoli di progetto	SI	NO	SI

Tabella 4.13: Costo di produzione totale ottenuto tramite le 3 diverse ottimizzazioni

Capitolo 5

Conclusioni e sviluppi futuri

5.1 Conclusioni

É stato dimostrato che seguendo la metodologia illustrata in questa tesi, è possibile giungere alla soluzione di problemi di ottimizzazione che non possono essere risolti con software commerciali. Ne sono un esempio le applicazioni presentate in precedenza, caratterizzate da funzioni obiettivo discontinue e dalla presenza di più discipline da tenere in considerazione. Basando il processo sulla realizzazione di un meta-modello predittivo, è possibile esplorare una grande quantità di possibili soluzioni in tempi molto più brevi rispetto alle tradizionali simulazioni CAE. Nel caso di problemi complessi, percorrendo un approccio tradizionale ci si potrebbe accontentare di una soluzione buona ma non ottima, senza esplorare a fondo il campo delle variabili. Per fare ciò sarebbe necessaria una quantità eccessiva di risorse, in termini di strumenti e soprattutto di tempistiche. L'interazione tra tecnologie di deep learning e algoritmo genetico consente di raggiungere soluzioni ottimali, riducendo il time-to-market (figura 5.1).

I vantaggi relativi all'utilizzo della metodologia proposta emergerebbero ancor più nel caso in si abbia già a disposizione uno storico di dati, legati a simulazioni effettuate in precedenza. Non avendo la necessità di costituire il database da zero, ciò permetterebbe di bypassare direttamente lo step 1 o di integrare all'interno del dataset le informazioni note, riducendo il numero di analisi da effettuare. In questo modo si abbatterebbero ancor di più i tempi di processo, in particolare quelli dedicati all'utilizzo della licenza del software.

In aggiunta ai benefici riguardo la precisione dei risultati e la riduzione dei tempi di processo, sfruttando una rete neurale per ogni componente si offre la possibilità di risolvere ottimizzazioni in cui vengono considerate diverse tipologie di variabili. Ne è una dimostrazione il problema presentato in precedenza, in cui, aggiungendo i materiali alle variabili di progetto, la scelta dell'ottimo è dettata dal giusto compromesso tra costo e caratteristiche meccaniche. Sono in via di sviluppo alcune varianti del problema relativo alla portiera del furgone, in modo da considerare l'impiego di più materiali, anche molto differenti rispetto ai due già utilizzati. Verrà presumibilmente modificato il caso di carico e allo stesso si

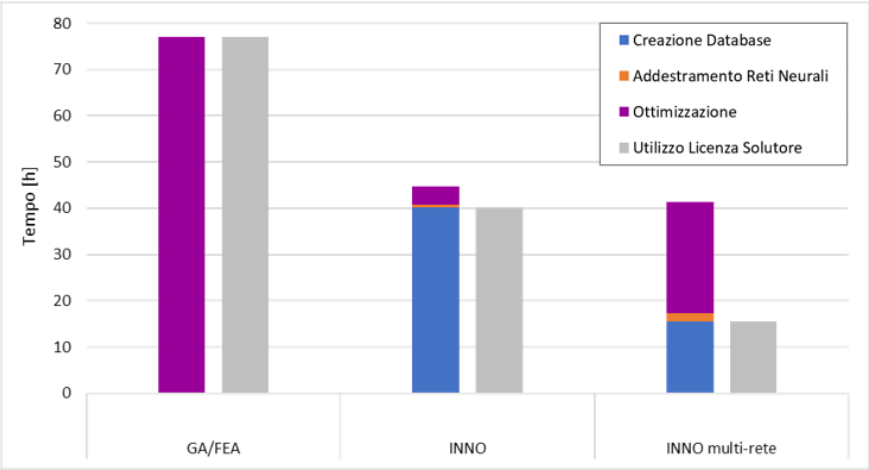

Figura 5.1: Confronto tra i tempi di processo relativi alle diverse metodologie

valuteranno altre risposte in aggiunta alle tensioni massime, come ad esempio spostamenti e frequenze proprie del sistema.

Inoltre, un'importante innovazione risiede nella possibilità di analizzare un sistema sotto diversi aspetti, considerando più discipline contemporaneamente nella risoluzione di problemi più complessi. Servendosi di più reti neurali, si può realizzare un meta-modello predittivo per ogni campo d'interesse, andando a sostituire ad esempio analisi termiche, strutturali, aerodinamiche, vibrazionali. In situazioni del genere si ha a che fare con ottimizzazioni multi-obiettivo, in cui ad ogni disciplina è associata una funzione da minimizzare. Sarà necessario riferirsi al concetto di ottimo paretiano, secondo cui le soluzioni ottime sono quelle che non vengono dominate, per le quali non esiste alcun punto che sia migliore per tutte le funzioni obiettivo (figura 5.2) [14].

5.2 Sviluppi futuri

Dopo aver illustrato i vantaggi che INNO multi-rete comporta, in quest'ultimo paragrafo verrà proposta una serie di modifiche ed aggiornamenti da attuare sul codice, al fine di migliorarne la precisione e ridurre i tempi di processo. Si prevede di intervenire su tutti gli step in cui è strutturato.

In primis, si esamineranno metodi statistici, o verrà creato un algoritmo apposito, per far sì che la definizione degli input di ogni modello segua una determinata distribuzione e non sia completamente randomica. Tale innovazione porterebbe all'ottenimento di un dataset più omogeneo, in cui ogni zona del dominio è descritta da una quantità molto simile di modelli. Inoltre, in questo modo, si eviterebbe la ridondanza di alcune configurazioni degli input e si ridurrebbero quindi i tempi relativi alla creazione del database.

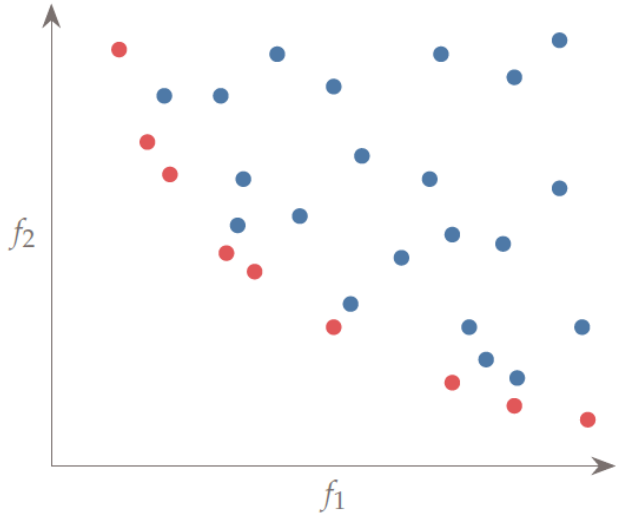

Figura 5.2: Ottimo paretiano

Allo stato attuale, il dominio da cui si ricavano gli input per l'addestramento delle reti coincide esattamente con il dominio dell'ottimizzazione. Nei problemi che presentano molte variabili di progetto, in corrispondenza dell'ottimo, alcune di esse raggiungono un valore pari al limite superiore o inferiore. Per permettere alle reti neurali di descrivere al meglio le zone in prossimità degli estremi, è conveniente estendere il campo degli input al di fuori del dominio di ottimizzazione. Da questa modifica ne trarrebbe vantaggio anche l'algoritmo, considerando che nella ricerca della soluzione ottima indaga anche zone al di fuori del dominio.

Per quanto riguarda l'addestramento delle reti neurali e del loop RETRAIN, nella versione attuale del codice, è richiesto di impostare inizialmente dei parametri, che rimangono costanti nel corso di tutto processo. I parametri in questione sono: numero di modelli iniziali, percentuale di incremento del numero di modelli, pendenza limite della curva che definisce l'errore in funzione del numero di modelli. Con l'obiettivo di generalizzare il più possibile la metodologia, non è concepibile definire dei valori che vadano bene per qualsiasi tipologia di problema. Per questo motivo è previsto lo studio di un algoritmo per il monitoraggio della funzione errore-modelli. La volontà è quella di effettuare incrementi del database in funzione dell'intensità con cui decresce la curva, prevedendo in un certo senso il suo andamento, tramite la valutazione della derivata volta per volta. Inoltre, l'addestramento ha una natura parzialmente randomica, a causa dell'inizializzazione di pesi e bias, e della selezione delle righe del dataset per il calcolo del gradiente. Ciò comporta che non sempre l'ultima rete del loop RETRAIN sia affetta da un errore minore rispetto alla penultima. Potrebbe essere interessante effettuare un ulteriore controllo per monitorare questo aspetto.

Così come il loop tra step 1 e step 2, anche il loop ZOOM è governato da parametri che rimangono costanti durante il processo, come ad esempio la percentuale di riduzione del dominio per ogni variabile di progetto e la dimensione minima del range. Ideare un algoritmo per far variare l'entità della riduzione, comporterebbe

vantaggi in termini di tempo; simultaneamente impedirebbe che una restrizione eccessiva possa escludere la soluzione dal dominio utilizzato per la creazione del dataset. Si potrebbe monitorare la variabilità di ogni elemento dell'array soluzione, e di conseguenza agire sui range, effettuando riduzioni più importanti per i valori che rimangono pressoché costanti nel corso delle varie iterazioni e invece essere più contenuti per quelli che oscillano maggiormente.

Inoltre, verrà effettuato uno studio approfondito sui parametri caratteristici dell'algoritmo di ottimizzazione utilizzato. Avendo una miglior conoscenza di ciò, sarà possibile modificare la dimensione della popolazione iniziale, la metodologia di ricombinazione tra gli individui, la percentuale di cross-over. In aggiunta, non è escluso l'utilizzo di altri algoritmi gradient-free, differenti da quello genetico.

Prossimamente, assieme all'implementazione delle modifiche appena presentate, si procederà con la risoluzione di problemi sempre più complessi, che coinvolgano almeno 3 discipline. Tramite l'utilizzo di tecniche di morphing e CAD parametrici si esplorerà anche il campo delle ottimizzazioni di forma.

Bibliografia

[1] *Algoritmo genetico*. URL: https://www.okpedia.it/algoritmo_genetico.

[2] Pragati Baheti. *12 types of neural network activation functions: how to choose?*

[3] Nicoletta Boldrini. *Reti neurali: cosa sono e a cosa servono*. 2022.

[4] Jason Brownlee. «Dropout regularization in deep learning models with keras». In: *Machine Learning Mastery* 20 (2016).

[5] Jason Brownlee. «Gentle introduction to the adam optimization algorithm for deep learning». In: *Machine Learning Mastery* 3 (2017).

[6] Jason Brownlee. «How to Choose Loss Functions When Training Deep Learning Neural Networks. 2019». In: *URL: https://machinelearningmastery. com/how-to-choose-loss-functions-whentraining-deep-learning-neural-networks* ().

[7] Jason Brownlee. «How to Code a Neural Network with Backpropagation In Python (from scratch)». In: *Code Algorithms From Scratch. URL: https://machinelearningmastery. com/implementbackpropagation-algorithm-scratch-python* (2016).

[8] Jason Brownlee. «Your First Deep Learning Project in Python with Keras Step-By-Step». In: *Machine Learning Mastery* (2020).

[9] Sanket Doshi. «Various optimization algorithms for training neural network». In: *Towards data science* 13 (2019).

[10] Pranath Kumar Gourishetty et al. «Global Optimization of a Turbine Design via Neural Networks and an Evolutionary Algorithm». In: *Optimization in Artificial Intelligence and Data Sciences*. Springer, 2022, pp. 259–267.

[11] Jeremy Jordan. «Setting the learning rate of your neural network». In: *URL: https://www. jeremyjordan. me/nn-learning-rate* (2018).

[12] *Keras*. URL: https://keras.io/.

[13] Jeffrey Larson, Matt Menickelly e Stefan M Wild. «Derivative-free optimization methods». In: *Acta Numerica* 28 (2019), pp. 287–404.

[14] Joaquim RRA Martins e Andrew Ning. *Engineering design optimization*. Cambridge University Press, 2021.

[15] Andrea Missinato. *Machine learning / Reti neurali demistificate*. 2018.

[16] Davide Nardini. *Train, Validation, Test: cosa sono e come si usano nel Machine Learning*. 2020.

[17] Jorge Nocedal e Stephen J Wright. *Numerical optimization*. Springer, 1999.

[18] Sushant Patrikar. «Batch, Mini Batch and Stochastic Gradient Descent. 2019». In: *URL: https://towardsdatascience. com/batch-mini-batch-stochastic-gradientdescent-7a62ecba642a* (2021).

[19] *Preprocessing data*. URL: `https://scikit-learn.org/stable/modules/preprocessing.html`.

[20] *Regolarizzazione*. URL: `https://alessiomorselli.github.io/DeepLearning/web/pages/teoria/regularization.html`.

[21] Ideen Sadrehaghiighi. *Gradient (derivative) based shape optimization with case studies*. Rapp. tecn. 2022.

[22] *scipy.optimize.differential$_e$volution*. URL: `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.differential_evolution.html`.

[23] Tarang Shah. «About train, validation and test sets in machine learning». In: *Towards Data Science* 6 (2017).

[24] Sagar Sharma. «Epoch vs batch size vs iterations». In: *Towards Data Science* 23 (2017).

[25] Iryna Sidorernko. *What Is a Dataset in Machine Learning: Sources, Features, Analysis*. 2021.

[26] SmartCAE. *Ottimizzazione strutturale*. URL: `https://www.smartcae.com/servizi/ottimizzazione-strutturale/`.

[27] Riccardo Talarico. *Introduzione alle reti neurali*. Tratto da YouTube: https://www.youtube.com/channel/UC6KzWKH-WyZkqLQNt6ptB9A/playlists. 2018.

[28] Giacomo Tontini. «Previsione di agenti inquinanti mediante reti neurali e ottimizzazione degli iperparametri attraverso grid search con Talos». In: ().

[29] Yugesh Verma. *A complete understanding of dense layers in neural networks*. 2021.