Alma Mater Studiorum · Università di Bologna

DIPARTIMENTO DI **INFORMATICA - SCIENZA E INGEGNERIA**
Laurea Magistrale in **Ingegneria Informatica**
Tesi Magistrale in **Intelligent Systems**

Supervised Learning with Graph Structured Data for Transprecision Computing

Relatore:
Chiar.ma Prof.ssa
MILANO MICHELA

Candidato:
LIVI FEDERICO

Correlatore:
Dr.
BORGHESI ANDREA

Sessione II
Anno Accademico 2018-2019

Indice

Abstract

Nell'era dell'Internet of things, dei Big Data e dell'industria 4.0, la crescente richiesta di risorse e strumenti atti ad elaborare la grande quantità di dati e di informazioni disponibili in ogni momento, ha posto l'attenzione su problemi oramai non più trascurabili inerenti al consumo di energia e ai costi che ne derivano.
Si tratta del cosiddetto *powerwall*, ovvero della difficoltà fisica dei macchinari di sostenere il consumo di potenza necessario per il processamento di moli di dati sempre più grandi e per l'esecuzione di task sempre più sofisticati.
Tra le nuove tecniche che si sono affermate negli ultimi anni per tentare di arginare questo problema è importante citare la cosiddetta *Transprecision Computing*, approccio che si impegna a migliorare il consumo dell'energia a discapito della precisione. Infatti, tramite la riduzione di bit di precisione nelle operazioni di floating point, è possibile ottenere una maggiore efficienza energetica ma anche una decrescita non lineare della precisione di computazione.
A seconda del dominio di applicazione, questo tradeoff può portare effettivamente ad importanti miglioramenti, ma purtroppo risulta ancora complesso trovare la precisione ottimale per tutte le variabili rispettando nel mentre un limite superiore relativo all'errore. In letteratura, questo problema è perciò affrontato utilizzando euristiche e metodologie che coinvolgono direttamente modelli di ottimizzazione e di machine learning.
Nel presente elaborato, si cerca di migliorare ulteriormente questi approcci, introducendo nuovi modelli di machine learning basati anche sull'analisi di relazioni complesse tra le variabili. In questo senso, si arriva anche ad esaminare tecniche che lavorano direttamente su dati strutturati a grafo, tramite lo studio di reti neurali più complesse, le cosiddette *graph convolutional networks*.

Introduzione

Il lavoro di tesi magistrale in esame è parte di un progetto europeo più ampio, OPRECOMP (Open TransPREcision COMPuting).
Si tratta di un progetto di ricerca della durata di 4 anni partecipante al Programma Quadro europeo per la Ricerca e l'Innovazione, Horizon 2020, che punta ad elaborare, dimostrare ed applicare il concetto di "approssimazione" al mondo tecnologico. In effetti, tale paradigma è presente innanzitutto nel modo di pensare dell'uomo, il quale pensa e decide sempre nell'ambito dell'approssimazione, in maniera da migliorare l'efficienza e il tempo generale di decision making.
Fino ad ora, nei sistemi di computazione tradizionali, ci si è sempre basati sull'inviolabile assunzione che ogni calcolo debba essere accurato.
Questo paradigma sfortunatamente è in declino e destinato a concludersi, in quanto la massima precisione dei risultati applicativi in generale tende ad andare ad impattare sul consumo di energia e sui relativi costi.
Approssimare significa dunque porre rimedio, ovvero "accontentarsi", in quei domini applicativi che lo permettono, di un risultato meno preciso ma che sia comunque significativo.
Il progetto OPRECOMP punta al principio essenziale secondo cui quasi ogni applicazione fa leva su nodi intermedi di calcolo, le quali precisioni sono completamente irrilevanti per l'utente finale che è maggiormente interessato all'affidabilità e alla validità del risultato finale.
OPRECOMP perciò nasce per dimostrare che la cosiddetta Transprecision Computing, ovvero la nuova astrazione di aprossimazione che si pone in alternativa al tradizionale paradigma di computazione, sarà effettivamente il modo in cui verranno pensati i nuovi sistemi. Inoltre, l'impatto delle approssimazioni intermedie sul risultato finale sarà del tutto controllabile e limitabile.
In definitiva, di tutto ciò ne trarranno maggiormente profitto tutti i campi di ricerca più vasti ed interessanti ad oggi quali IoT, Big Data Analisys e Deep Learning.

Nei prossimi capitoli verrà prima presentato lo stato dell'arte in ambito Transpecision Computing e tutti gli strumenti operativi ad esso associati, per poi esaminare diversi modelli atti a migliorarne ancora di più l'efficienza e l'affidabilità dei risultati.
Si esaminerà innanzitutto l'attuale piattaforma di Active Empirical Model Learning [1],la quale, brevemente, consente attualmente di utilizzare le conoscenze apprese tramite una rete neurale, allenata in maniera attiva, ed un classificatore per la risoluzione di un problema di ottimizzazione.
Utilizzando come campioni operazioni di tipo floating point, si va nello specifico ad individuare quale sia il numero minimo di bit da associare ad ognuna delle variabili in gioco durante l'esecuzione delle suddette operazioni. Data la complessità del problema in esame, verranno studiati ed analizzati diversi approcci ed euristiche inerenti al machine learning.
Più nello specifico, nei capitoli successivi, partendo da lavori e risultati già esistenti in questo ambito, verranno testati diversi modelli di regressori e reti neurali per tentare di gestire al meglio il tradeoff tra qualità finale del risultato e precisioni in bit assegnate alle variabili.
In particolare poi, verrano considerate anche soluzioni più complesse tra le quali modelli di deep learning e graph convolutional networks.
Su queste ultime in particolare verrà posta maggiormente l'attenzione nella parte finale dell'elaborato, in quanto si tratta di un approccio del tutto nuovo: partendo da relazioni tra le variabili non considerate precedentemente, si crea una base di dati a grafo, sfruttata poi per costruirvi sopra i vari modelli di GCN.
Infine, negli ultimi capitoli, verranno esaminati i diversi risultati ottenuti sfruttando le diverse soluzioni operative, evidenziandone svantaggi e vantaggi di ognuna ed effettuandone una comparazione riassuntiva.

[1]Come si vedrà successivamente, il lavoro di Lombardi et.al. propone un modello di Empirical Model Learning che ha lo scopo di integrare componenti di decision making e di ottimizzazione combinatoria.

1 Transprecision Computing

Come accennato nell'introduzione, OPRECOMP punta a demolire l'ultra conservativa astrazione di computazione precisa in favore di una concezione del tutto nuova, più efficiente e veloce, chiamata Transprecision Computing.
Tale nuova concezione affonda le sue radici nell'intuizione dello sfruttamento dell'approssimazione per incrementare l'efficienza energetica.
Infatti, la Transprecision Computing si basa sul concetto di approximate computing[7][8], ovvero quel vasto insieme di tecniche atte ad individuare quale sia il massimo livello di approssimazione tollerabile.
Inoltre, tale nuovo paradigma si spinge oltre, poiché [2]:

- Gestisce l'approssimazione nello spazio e nel tempo attraverso molteplici controlli sia software che hardware;
- Non implica la riduzione della precisione a livello applicativo, anche se rimane comunque possibile rilassarne dei vincoli per ottenere maggiori benefici.

Attraverso questa nuova astrazione, si punta a costruire e a dimostrare i vantaggi di un framework basato sull'approssimazione dal range dei mW a quello dei MW, spaziando così nove ordini di grandezza [1].
In definitiva, l'obbiettivo della Transprecision Computing è di fatto operare in modo da avere un miglioramento di almeno un ordine di grandezza nell'efficienza energetica [2], dimostrando inoltre che tale idea può essere sfruttata per trarne dei vantaggi in una grande numero di scenari applicativi che spaziano nei domini dell'IoT, dei Big Data, del Deep Learning e delle simulazioni HPC e che riguardano ambiti sia hardware che software.

Per quanto riguarda il primo, è ormai risaputo che in tante applicazioni il problema non è la computazione in sè, ma la limitatezza della larghezza di banda di memoria che gioca un ruolo cruciale nelle architetture di computazione più avanzate.

Al giorno d'oggi, le memorie più importanti ed utilizzate sono sicuramente le Dynamic Random Access Memories (DRAMs) che contribuiscono però largamente al consumo di potenza generale e alle performance di un sistema. Ad esempio, è stato dimostrato che nel natural language processing, più dell'80% del consumo totale di energia proviene dalle DRAMs e solo il 18% dalla computazione in sè.
Nell'ambito della Transprecision Computing, si affronta questa problematica introducendo una versione approssimata delle DRAMs e si sfruttano le conoscenze dei controller delle stesse per orchestrare e gestire al meglio gli accessi alle memorie. Tutto ciò può portare ad incrementi significativi dell'efficienza energetica e alla massimizzazione della banda di memoria.

Con tale elaborato, verrà però considerata l'applicazione della Transprecision Computing nell'altro livello, quello software, e più specificamente andando ad agire sulla rappresentazione dei dati utilizzati per la computazione. In effetti, nelle moderne piattaforme a basso consumo di energia, l'esecuzione di operazioni floating-point (FP) emerge come una delle maggiori cause di consumo di energia nelle applicazioni a computazione intensiva. Esperimenti e test scientifici mostrano infatti che più del 50% dell'energia consumata durante l'elaborazione e la computazione dipende effettivamente dal calcolo FP [3].
Di conseguenza, l'adozione di formati FP che richiedono un minor numero di bit di precisione, è un'interessante opportunità per ridurre il consumo energetico, poiché consente di semplificare i circuiti aritmetici e di diminuire il consumo di memoria richiesto per trasferire dati dai registri.
Da un punto di vista puramente teorico poi, il tuning e la modifica a livello di operazioni FP è perfettamente in linea con la materia della Transprecision Computing in quanto tale, poiché permette appunto di controllarne l'approssimazione considerando però anche specifici vincoli di qualità del risultato.
Per questo motivo, nel prossimo sottocapitolo si introduce innanzitutto la rappresentazione Floating Point dei dati per poi passare alla descrizione di una piattaforma ad-hoc nata per interagirci al meglio.

1.1 Computazione con numeri FP

La maggior parte delle applicazioni che trattano di computazioni numeriche adottano i floating-point come tipo di dato, ovvero una rappresentazione approssimata dei numeri reali e di elaborazione dei dati.
Lo IEEE Standard per l'aritmetica floating-point (IEEE 754) è uno standard tecnico per la computazione FP che è stato stabilito nel 1985 dall'Institute of Electrical and Electronics Engineers (IEEE).
Lo standard nasce per risolvere diversi problemi emersi nelle diverse implementazioni FP che hanno di fatto reso difficile la portabilità e la scalabilità di tali rappresentazioni in diversi ambienti. Nello specifico, l'IEE Standard 754 è la più comune rappresentazione FP al giorno d'oggi per quanto riguarda i numeri reali in tutti i calcolatori, inclusi PC, Macs e la maggior parte delle piattaforme Unix.
Di conseguenza, un numero reale può essere inteso come floating-point in diversi modi, ma quello relativo allo IEEE 754 è sicuramente il più efficiente nella maggior parte dei casi ed è il seguente:

$$n = \pm m \cdot b^{\pm e}$$

Tale rappresentazione è costituita da:

- m ovvero la mantissa, un numero in valore assoluto che rappresenta le cifre significative;
- b ovvero la base;
- e ovvero l'esponente cui elevare la base b (è un numero relativo).

Esiste inoltre una forma normalizzata, in cui la mantissa ha la prima cifra significativa (diversa da zero) subito dopo la virgola.
Occorre inoltre fissare un numero di bit N_m per il valore assoluto della mantissa, e un numero di bit N_e per l'esponente in complemento a 2 come mostrato più specificamente dall'immagine sottostante:

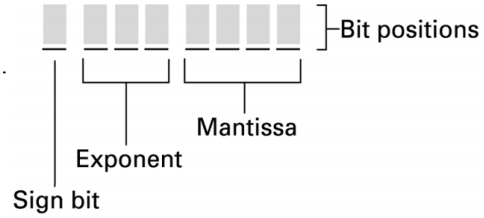


Figura 1.1: Rappresentazione di un numero FP

Le precisioni sostanzialmente più diffuse ed utilizzate sono quattro [3]:

- Singola precisione, 32 bit;
- Doppia precisione, 64 bit;
- Estesa-Doppia precisione;
- Quadrupla precisione;

Al giorno d'oggi, la pratica generale seguita maggiormente è assegnare la massima precisione (64bit o più) a tutte le variabili di un programma per garantirne la precisione assoluta dei risultati. Questo logicamente, come spiegato precedentemente, si contrappone fortemente all'idea di mantenere una certa efficienza energetica e perciò nel tempo i ricercatori hanno studiato tecniche software focalizzate sulla riduzione dei formati FP.
In quest'ottica, l'IEEE ha quindi introdotto una quinta tipologia di precisione, la cosiddetta binary16, costituita da solamente 16 bit.
Tra le ricerche effettuate in merito, è importante citare il lavoro svolto da Tavaglini et. al.[3], i quali hanno realizzato una piattaforma per l'esecuzione di operazioni FP che supporti la Transprecision Computing, dimostrandone allo stesso tempo la validità, FlexFloat.
Si tratta di un software open-source sviluppato definendo una Application Programming Interface (API) per supportare molteplici (standard e non) formati FP e operazioni aritmetiche tra di essi.
A differenza di altre librerie simili tra cui PROMISE[5] e fpPrecisionTuning[6], Flexfloat:

1. abilita un'accurata manipolazione dei formati andando ad agire direttamente sui campi della mantissa e dell'esponente;
2. utilizza una metodologia di emulazione che fa leva su tipi nativi per ridurre significativamente il tempo richiesto per fare il tuning della precisione;
3. fornisce features fondamentali nel campo della Transprecision Computing, come statistiche a runtime sulle operazioni aritmetiche effettuate e la possibilità di tracciare gli errori tramite opportune callbacks definite dall'utente.

In particolare, è importante prendere in considerazione tale tool nell'elaborato seguente, dato che con esso vengono proposti due formati FP estesi di particolare rilevanza:

- Binary16alt, ovvero un formato FP a 16 bit complementare a quello proposto dallo standard IEEE 745;
- Binary8, ovvero un formato FP a 8 bit che prevede una bassa precisione in cui si utilizzano solo 3 bit per la rappresentazione della mantissa.

Con questi nuovi formati si riesce a dare un significativo miglioramento alle tecniche inerenti la Trasprecision Computing in quanto lavorare con essi si è dimostrato essere la scelta giusta nella strada del miglioramento dell'efficienza energetica. In effetti, utilizzando un numero di bit ridotto, le operazioni sono logicamente più veloci di quelle eseguite su dati ad alta precisione. A questo proposito, si è osservato sperimentalmente che in alcuni domini utilizzare una computazione di questo genere permette di ridurre in media dal 20 al 52% il tempo di esecuzione e dal 22 al 60% il consumo di energia per una grande varietà di benchmark effettuati[3]. Infine, risultati sperimentali mostrano inoltre che FlexFloat performa meglio di tutte le altre librerie inerenti al tuning della precisione, altro fattore che ne ha spinto l'adozione in questo elaborato.

1.2 Stato dell'arte

Con la possibilità introdotta da FlexFloat di eseguire il tuning delle precisioni delle variabili in un programma, risulta chiaro che la vera sfida ora si sposta sul trovare il setup migliore, in modo da avere sì una buona approssimazione ma anche assicurarne una certa qualità.
Tutto ciò è inscrivibile in un problema di ottimizzazione e risolvibile con tecniche mirate come la Mathematical Programming (MP) o la Costraint Programming (CP).
Infatti, l'idea è quella di andare a ricercare il numero minimo di bit che possono essere assegnati ad ogni variabile del programma senza incorrere in un errore di computazione più grande di un errore target prestabilito [9]. Più in generale, vengono considerate applicazioni numeriche, chiamate benchmark[1], dove molteplici variabili FP contribuiscono alla computazione dell'output finale, dato un certo input set (ovvero un vettore o una matrice di valori FP).
Il numero delle variabili con una precisione controllabile in un benchmark B è n_B^{var} e assegnare una precisione significa di fatto decidere il numero di bit per l'esponente e la mantissa del valore FP (attualmente lo stato

[1]Più dettagli su tale argomento si trovano nel sottocapitolo successivo

dell'arte prevede che sia lo stesso valore per entrambi).
In generale quindi, l'errore che ne deriva corrisponde alla riduzione della qualità dell'output a causa del tuning della precisione delle variabili.
Formalmente, se l'output è vettore costituito da N_O valori, O rappresenta il risultato con il tuning della precisione, $\hat{O}$ indica il risultato a precisione massima e l'errore E è calcolato come:

$$E = \max_i \frac{(o_i - \hat{o}_i)^2}{\hat{o}_i^2} \qquad i \in [1, N_O]$$

Il problema associato a tutto ciò è che occorre esprimere la relazione non lineare tra la precisione e l'errore computazionale in una forma analitica, che non è affatto banale.
Inoltre, la complessità del problema viene incrementata all'aumentare delle variabili in gioco, a causa degli effetti non trascurabili prodotti dall'interazione tra di esse. Più in particolare, lo spazio precisione-errore è molto complesso, non regolare e con molti picchi (ottimi locali).
Inoltre, la funzione che lega la precisione all'errore è fortemente non lineare e anti-monotona: per esempio, aumentare la precisione di un set di variabili può portare ad errori ancora più grandi di quelli dovuti ad una configurazione con un numero minore di bit.
Questo accade a causa di una serie di fattori: l'impatto delle operazioni di arrotondamento, l'instabilità numerica dei benchmark considerati, e il legame di codipendenza tra diverse variabili (in questo senso aumentare la precisione di una variabile può essere controproducente se poi viene assegnata ad una variabile con una precisione inferiore).

Di conseguenza, un'analisi statica di tali effetti in una maniera del tutto tradizionale non è sicuramente la strada giusta verso la soluzione.
Invece, un approccio diverso è quello legato all'imparare tali legami tra le variabili in gioco invece che esprimerli con chiarezza a priori.
Si utilizzano perciò dei modelli di machine learning e una volta creata questa base di conoscenza, l'idea è quella di inserire il tutto in un modello di ottimizzazione per risolvere finalmente il problema.
Questa è l'idea generale alla base dell'approccio di Empirical Model Learning (EML)[10, 11], tecnica per abilitare quindi l'ottimizzazione combinatoria su sistemi reali complessi.
Essa propone un modello di ottimizzazione basato su tre componenti:

1. un modello MP;
2. un modello ML per predire l'errore associato ad una configurazione di precisione delle variabili;

3. un modello ML per classificare le configurazioni in due macro classi a seconda dell'errore associato.

In particolare, i due modelli ML sono inclusi nel modello MP per rappresentare la conoscenza sulla relazione tra la precisione delle variabili e l'errore di output.
Il modello MP cerca la configurazione di bit ottimale per un'applicazione basandosi sulle predizioni dei due modelli di machine learning: per stabilire la qualità della configurazione occorre in particolare eseguire il benchmark con la corrispettiva precisione utilizzando FlexFloat, il quale assegna ad ogni variabile un numero specifico di bit.
Come già motivato precedentemente, il compito dei modelli ML è molto difficile, poiché l'obiettivo è in pratica imparare una funzione matematica molto complessa.
Di conseguenza, la soluzione trovata dal modello MP potrebbe essere sbagliata, ovvero non rispettare il vincolo dell'errore target, a causa del gap troppo grande tra errore reale e stimato.
Per risolvere questo problema si introduce perciò un loop di raffinamento: si testa inizialmente la soluzione MP eseguendo il benchmark con la specifica precisione e se la soluzione è sbagliata se ne trova una nuova. Quella errata (ovvero la configurazione più l'errore) è aggiunta al training set dei modelli ML (di cui poi viene rieseguito il training) e tagliata fuori dal set di possibili soluzioni del problema di MP (attraverso dei vincoli). Un nuovo modello MP è poi costruito basandosi sui modelli ML raffinati e una nuova ricerca comincia. Questo loop continua finché non viene trovata una possibile soluzione (non per forza quella ottimale).
Tutto il processo è presentato in maniera riassuntiva dalla figura sottostante:

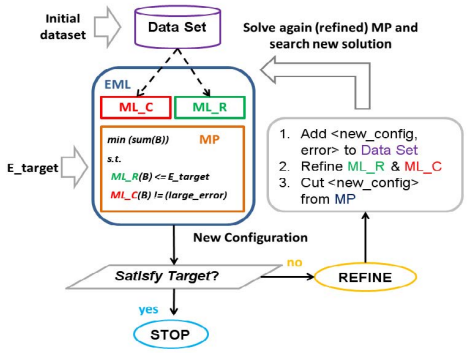


Figura 1.2: Schema dell'approccio EML proposto[9]

1.3 Benchmark

Allo stato dell'arte attuale, i test del modello EML utilizzato vengono effettuati con l'ausilio di un particolare subset di applicazioni studiate nel contesto della Transprecision Computing, scelte perché presentano casi di studio differenti tra di loro e poichè incapsulano diversi pattern di computazione.
In particolare, sono stati scelti i seguenti benchmark, a cardinalità crescente delle variabili:

- *FWT*, ovvero la *Fast Walsh Transform* per i vettori reali, dal dominio dell'algebra lineare avanzata (la cardinalità delle variabili è 2);

- *saxpy*, ovvero un'addizione vettoriale dal dominio dell'algebra lineare semplice che moltiplica un vettore di input, X, per uno scalare a e somma tutto ad un secondo vettore, Y.
  Formalmente:
  $$y_i = ax_i + y_i$$
  La cardinalità delle variabili è 3;

- *convolution*, ovvero la convoluzione di una matrice con un kernel 11x11 (la cardinalità delle variabili è 4);

- *dwt*, ovvero la *Discrete Wavelet Transform*, dalla teoria dei segnali (la cardinalità delle variabili è 7);

- *correlation*, ovvero il calcolo della matrice di correlazione dell'input, dal dominio del Data Mining (la cardinalità delle variabili è 7);

- *BlackScholes*, ovvero l'operazione in ambito finanziario che stima il prezzo per un set di opzioni applicando l'equazione differenziale parziale di Black-Scholes (la cardinalità delle variabili è 15);

- *Jacobi*, ovvero un algoritmo iterativo utilizzato per determinare la soluzione di un sistema di equazioni linerari diagonalmente dominanti (la cardinalità delle variabili è 25).

Ognuno dei benchmark descritti si occupa di operare tra vettori di dimensioni diverse.
Considerando la tabella sottostante rappresentante un ipotetico dataset di DWT e chiamata configurazione un set di precisioni in bit, perciò se ne deduce che il numero che si associa ad ognuna delle variabili non è l'effettivo valore di esse, ma appunto il numero di bit utilizzati per descriverle:

	var0	var1	var2	var3	var4	var5	var6	errds
config0	11	25	44	23	40	13	18	0.208
config1	15	32	7	18	15	23	42	0.309
config2	12	5	45	44	43	25	16	0.687
config3	42	22	36	18	19	40	8	0.268

Nel presente elaborato verranno utilizzati solamente i benchmark più significativi, ovvero Convolution, Correlation, DWT, BlackScholes e Jacobi.
In effetti essi, a cardinalità crescente di variabili, permetteranno già di avere un quadro abbastanza completo dei vari test.

1.4 Grafo delle dipendenze

Riprendendo quanto detto nei sottocapitoli precedenti, si deduce che il modello MP assegna un valore di precisione ad ogni variabile nel benchmark e minimizza il numero totale di bit, rispettando il limite superiore sull'errore.
In particolare, nel problema di ottimizzazione che ne risulta, sono presenti molteplici vincoli da risolvere dato che un modello ML può richiederne fino a centinaia o addirittura migliaia. Tra questi, una tipologia in particolare deriva dal cosiddetto grafo delle dipendenze del benchmark, che specifica come le variabili del programma sono correlate tra di loro.
Per esempio, considerando l'espressione $V_1 = V_2 + V_3$, si sa che in fase di compilazione FlexFloat imporrà che la precisione di V_1 sia minore o uguale di quella data dalla somma delle precisioni di V_2 e V_3.
In altre parole, la precisione della variabile a destra dell'uguale dipende da quella delle variabili a sinistra.

Per realizzare questo comportamento, verranno utilizzate a runtime quattro variabili corrispondenti, $x_i, i \in [1, 4]$. In particolare, la variabile x_4, è una variabile temporale alla quale verranno convertite e assegnate i valori delle precisioni corrispondenti a V_2 e a V_3.
Di fatto, ogni variabile può essere intesa come un nodo in un grafo di dipendenze in cui le relazioni tra di esse vengono rappresentate con archi direzionati, come mostrato in figura:

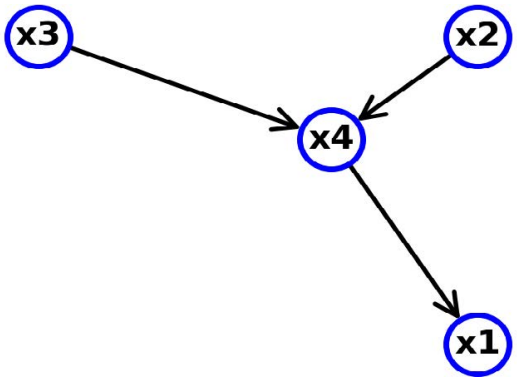


Figura 1.3: Esempio di grafo delle dipendenze[9]

Un arco che entra in un nodo significa che la precisione della variabile di partenza è collegata a quella della variabile di arrivo.
Dal grafo proposto, si possono ottenere quindi vincoli aggiuntivi che riescono a limitare fortemente lo spazio di ricerca, andando in questo modo a ridurre in maniera consistente il tempo necessario a trovare una soluzione.
Ci si concentra su due tipi di relazione:

1. assegnamento, come ad esempio $x_4 \rightarrow x_1$ in cui si impone che la variabile x_4 abbia una precisione minore o uguale a quella di x_1, ovvero:
$$x_4 \leq x_1$$
2. cast, come ad esempio $x_2, x_3 \rightarrow x_4$. Ciò implica che il risultato dell'espressione che coinvolge le due variabili x_2 e x_3 deve convergere alla precisione associata alla variabile addizionale x_4.
Più formalmente, si implica che la variabile addizionale abbia una precisione minore o uguale alla minima precisione degli operandi coinvolti nell'espressione, ovvero:
$$x_4 = min(x_2, x_3)$$

In generale inoltre, sono possibili anche operazioni più complesse e che coinvolgono anche un numero molteplice di variabili.

1.5 Approccio proposto

Una delle limitazioni principali del modello di computazione attuale descritto nei sottocapitoli precedenti è l'impiego di una grande quantità di dati necessaria per un corretto training del regressore, componente poi integrato nella pratica.
Per ovviare a tale problema, nel lavoro di Viarchi[12] si propone un metodo di learning alternativo, l'Active Learning, grazie al quale è possibile utilizzare per il training del regressore un dataset a dimensionalità ridotta, il quale viene ripopolato durante i vari step esecutivi con alcuni degli esempi prodotti dal regressore stesso.
Invece, con la seguente tesi si propone un possibile miglioramento a monte del problema, andando a ricercare e a studiare modelli di ML più precisi per predire gli errori associati alle configurazioni.
In effetti, fino ad ora sono stati utilizzati semplici modelli di reti neurali nel modello ML inserito poi dentro al problema di ottimizzazione, ma ci si è resi conto che essi non sono sufficienti a fornire alta precisione nei risultati. Inoltre, con tale elaborato, si estende anche il modello ML attuale per essere compatibile con altri input set, anche se ancora nella pratica esso esegue una predizione per un particolare input set (per ottenere risultati con altri input set, occorre allenare un modello differente). Questa assunzione è stata fatta perché si tratta di un problema nuovo che viene affrontato in letteratura in maniera analogo (cioè un input set per volta). Un altro approccio ideale sarebbe ad esempio considerare tutti gli input set per allenare lo stesso modello, ma rimane un problema aperto.
Nei capitoli successivi, si porrà perciò l'attenzione allo studio e all'implementazione di modelli di ML a difficoltà crescente, per verificare quali siano i vantaggi e gli svantaggi di ogni soluzione ed effettuare una comparazione tra di esse. In questo senso, si andrà prima ad utilizzare semplici regressori, per poi passare a modelli di reti neurali via via più complesse fino a spaziare anche alle graph convolutional networks.
Per quanto riguarda quest'ultima tipologia, si fa inoltre leva sul grafo delle dipendenze per tentare di cercare un ulteriore miglioramento della precisione del modello ML. In effetti, tale grafo è perfettamente compatibile con questa tipologia di rete neurale, introdotta per la prima volta da Kipf [13] e ripresa poi in lavori successivi, come Spektral, framework ad-hoc ad opera di Grattarola [32].

2 Machine learning

Il machine learning è quella branchia dell'intelligenza artificiale intesa come abilità dei calcolatori di apprendere senza essere stati esplicitamente e preventivamente programmati. In effetti, è un approccio totalmente diverso da quello più tradizionale, in cui tutto deve essere definito perfettamente affinché avvenga la corretta computazione.
Di fatto, si tratta di una disciplina molto vasta che prevede differenti tecniche e strumenti ed inoltre, le diverse modalità di apprendimento e sviluppo degli algoritmi danno vita ad altrettante possibilità di utilizzo che allargano ancora di più il campo di applicazione dell'apprendimento automatico.
A seconda del tipo di algoritmo utilizzato per permettere l'apprendimento alla macchina, ossia a seconda delle modalità con cui la macchina impara ed accumula dati e informazioni, si possono suddividere quattro differenti sistemi di apprendimento automatico:

- Supervisionato: consiste nel fornire al sistema una serie di nozioni specifiche e codificate, ossia di esempi che permettono di costruire un vero e proprio database di informazioni.
  In questo modo, quando la macchina si trova di fronte ad un problema, non dovrà fare altro che attingere alle esperienze inserite nel proprio sistema, analizzarle, e decidere quale risposta dare sulla base di qualcosa di già codificato. In base allo stimolo dato alla macchina, infatti quest'ultima risponderà andando a considerare le esperienze passate restituendo l'elaborazione del dato più attinente possibile.
  Gli algoritmi che fanno uso di apprendimento supervisionato vengono utilizzati in molti settori, da quello medico a quello di identificazione vocale: essi, infatti, hanno la capacità di effettuare ipotesi induttive, ossia ipotesi che possono essere ottenute scansionando una serie di problemi specifici per ottenere una soluzione idonea ad un problema di tipo generale. Più in dettaglio, l'allenamento supervisionato è tipicamente effettuato quando occorre mappare un

qualsiasi input in un dato di output che coincide con un etichetta (in questo caso si parla di classificazione), oppure quando occorre mappare un dato in input in un dato di output a valori continui (in questo caso si parla di regressione).
Algoritmi comuni nell'apprendimento supervisionato sono la logistic regression[14], i classificatori naive bayes[15], le support vector machines[16], le reti neurali[14] e le tecniche di random forest[17].
Dunque, sia nella classificazione che nella regressione, l'obbiettivo è quello di trovare relazioni specifiche nei dati in input al fine di produrre correttamente dati in output;

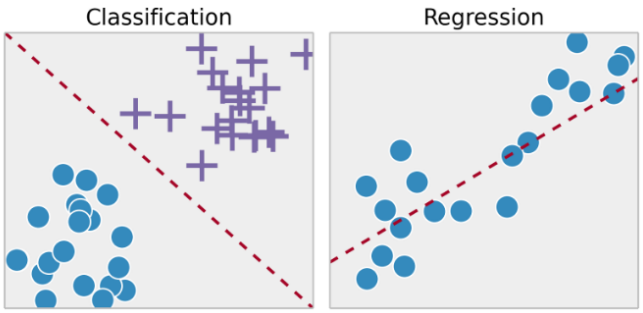


Figura 2.1: Classification vs regression[18]

- Non supervisionato: prevede la possibilità da parte della macchina di attingere a determinate informazioni senza avere alcun esempio del loro utilizzo e, quindi, senza avere conoscenza dei risultati attesi a seconda della scelta effettuata.
  Dovrà essere la macchina stessa, quindi, a catalogare tutte le informazioni in proprio possesso, organizzarle ed imparare il loro significato, il loro utilizzo e, soprattutto, il risultato a cui esse portano.
  L'apprendimento senza supervisione offre maggiore libertà di scelta alla macchina che dovrà organizzare le informazioni in maniera intelligente e imparare quali sono i risultati migliori per le differenti situazioni che si presentano.
  Il task principale eseguito con questa tecnica è sicuramente quello legato al clustering[19], ambito in cui si vuole imparare la struttura dei dati senza l'utilizzo di etichette.
  Si tratta di un approccio molto utile nell'analisi dei dati, in quanto permette di identificare automaticamente delle strutture e dei pattern ricorrenti.
  In effetti, esistono situazioni dove è praticamente impossibile per

l'uomo cercare trend dall'analisi dei dati, ed è proprio qui che le tecniche di clustering si rivelano vincenti;

- Semi-Supervisionato: è un modello ibrido in cui viene fornito un set di dati incompleti per l'allenamento.
  Si parla perciò sia di apprendimento supervisionato in quanto alcuni input sono dotati di output di esempio, sia di apprendimento non supervisionato, in quanto alcuni dati di input non hanno nessuna etichetta di riferimento;

- Per rinforzo: si tratta del sistema di apprendimento più complesso, che prevede che la macchina sia dotata di sistemi e strumenti in grado di migliorare il proprio apprendimento e, soprattutto, di comprendere le caratteristiche dell'ambiente circostante.
  In questo caso, quindi, alla macchina vengono forniti una serie di elementi di supporto che permettono di rilevare quanto avviene nell'ambiente circostante ed effettuare scelte per un migliore adattamento all'ambiente intorno a loro. Il sistema quindi interagisce in maniera del tutto dinamica imparando dagli errori, che di fatto infliggono delle punizioni, al fine di arrivare ad un obbiettivo, ottenendo conseguentemente una ricompensa.

In tutti i casi sopra citati, la qualità dei dati su cui lavorano i tali algoritmi è un fattore essenziale.
In effetti, il livello della complessità del modello di machine learning risultante dipenderà nella pratica dalla complessità della funzione che lega l'input all'output, la quale verrà imparata più o meno velocemente e più o meno efficientemente in relazione alla qualità dei dati.
Con una quantità di dati molto esigua su cui lavorare, che in particolare non copre tutti gli scenari possibili, capita infatti che la funzione scoperta sarà ottima nel descrivere quegli specifici dati, ma avrà molte difficoltà agendo su nuovi. Si tratta del cosiddetto fenomeno dell'overfitting, che si contrappone all'underfitting, dove il modello non riesce a descrivere per niente la variabilità dei dati (si consideri a tal proposito la figura 2.2).
Inoltre, per aumentare e gestire la qualità dei dati su cui poi dovrà lavorare un modello, vengono effettuate molteplici operazioni iniziali.
Tra queste troviamo, in ordine di processamento:

1. aggregazione: consiste nel combinare due o più attributi relativi ad uno specifico dato in modo da ridurne la quantità (come ad esempio aggregare anno, mese e giorno in un unico dato).

Inoltre si può operare anche andando a modificarne la scala, producendo di fatto dati aggregati più stabili in quanto tendenti ad avere meno variabilità;

2. campionamento: tecnica che consente di ridurre in maniera consistente la quantità dei dati da processare andando di fatto ad analizzare solamente parti del dataset inferendo poi le caratteristiche della versione estesa. Per questo motivo, in questa fase è strettamente necessario che il campione da selezionare sia rappresentativo, ovvero che abbia in generale le stesse proprietà del dataset originale;

3. riduzione della dimensionalità: quando la dimensionalità di un dataset è molto alta, l'occupazione della memoria diventa sostanzialmente sparsa e risulta perciò molto complesso lavorare sui dati. Tramite metodi matematici, come la PCA (Principal component analysis[21]) si può far fronte a questo problema in maniera da ridurre il rumore generale del dataset e la complessità temporale e spaziale richiesta per lavorarci;

4. selezione di un subset di attributi: è una tecnica mirata a ridurre la dimensionalità che procede andando ad agire sugli attributi ridondanti, ovvero che sono contenuti in altre porzioni dei dati, o su quelli irrilevanti, ovvero che non contengono nessuna informazione utile per l'analisi;

5. creazione di attributi: modalità in cui vengono creati nuovi elementi che riescono a descrivere meglio certe caratteristiche dei dati;

6. trasformazione degli attributi dei dati: si tratta di un insieme di tecniche matematiche per convertire i dati in altri opportuni formati con cui l'algoritmo di machine learning selezionato può lavorare meglio[1].

Infine, tali dati elaborati vengono suddivisi dall'algoritmo di machine learning in tre diverse categorie ed utilizzati per effettivamente avviare il processo di learning:

1. training set ovvero i dati utilizzati nella fase di addestramento del modello;

[1] A questo proposito si rimanda il lettore al sottocapitolo 4.2, in cui si parla di normalizzazione e standardizzazione, operazioni di trasformazione dei dati effettivamente utilizzati in questo elaborato.

2. validation set ovvero i dati utilizzati per la calibrazione di alcuni particolari parametri specifici dell'algoritmo (i cosiddetti iperparametri);

3. test set: i dati utilizzati per la valutazione delle performance del sistema.

Di seguito, in particolare verranno analizzate in maniera teorica le tecniche di machine learning utilizzate nel presente lavoro di tesi, i regressori e le reti neurali.

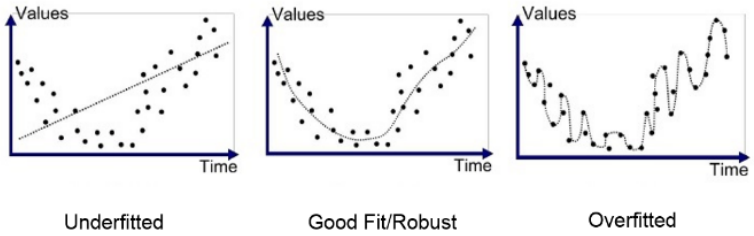


Figura 2.2: Esempi delle varie tipologie di fitting dei dati[20]

2.1 Regressori

I modelli di machine learning adibiti alla predizione di valori continui sono chiamati regressori e, come approfondito precedentemente, sono categorizzati nelle tecniche di apprendimento supervisionato, in quanto si basano su degli esempi, dei dati in ingresso, per produrre delle predizioni.
Più formalmente, l'obiettivo dei modelli di regressione è di approssimare una funzione f dai valori di input X a valori continui in output y.
Per valori continui si intendono valori reali, come interi o floating-point che rappresentano quantità e grandezze.
A tal proposito, un esempio classico di questa tipologia di modello può essere descritto attraverso la predizione del prezzo di un'abitazione date in ingresso al sistema le sue caratteristiche principali quali grandezza, posizione, numero delle stanze, ecc.
Esistono varie tipologie di regressione, descritte in breve qui di seguito e al giorno d'oggi frutto di ampi studi e ricerche [22][23]:

- Regressione lineare: si tratta del più comune ed interessante tipo di regressione in cui si predice una variabile target Y sulla base di una variabile in input X.
  Inoltre, deve esistere una relazione di linearità tra le variabili di input e quelle di output (concetto dal quale prende il nome il modello), ovvero deve sussistere quella relazione matematica che rende possibile la rappresentazione della funzione tramite una retta.
  In particolare, in un modello di regressione lineare semplice, il numero delle variabili indipendenti (ovvero Y) è uno.
  La rappresentazione di una possibile relazione di linearità tra le variabili di una funzione f, può essere descritta con la seguente figura:

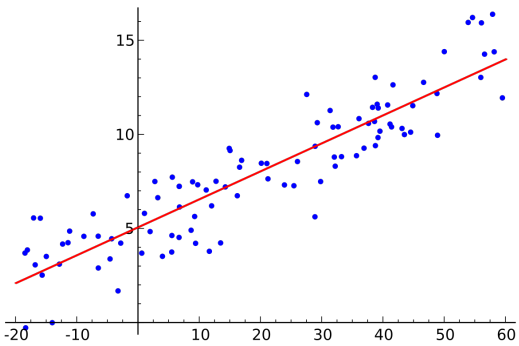


Figura 2.3: Modello di regressione lineare semplice

In particolare, la retta disegnata in rosso è quella che rappresenta in maniera migliore tutti i punti descritti dalla funzione f che ad esempio può essere descritta come:

$$Y = a + bX$$

Questo modello di machine learning quindi sarà incaricato di imparare i coefficienti a e b durante il training del regressore stesso.
In altre parole, occorre trovare quella retta che approssimi nella maniera più precisa possibile tutti i punti della funzione, ovvero tutti i dati del dominio. Per fare ciò, si utilizza una cosiddetta funzione di costo inserita in un problema in cui occorre minimizzare l'errore tra il valore reale e quello predetto dal modello.
La funzione di costo più comune è sicuramente la Mean Squared Error (MSE) che indica la media aritmetica dei quadrati della differenza tra i valori predetti e i valori reali[2].
Per minimizzare tale valore si utilizza poi la discesa del gradiente, metodo in cui intuitivamente si parte da valori casuali iniziali dei coefficienti che poi vengono modificati iterativamente al fine di ridurne il costo: tale processo viene poi interrotto nel momento in cui si trova un valore di minimo per la funzione;

- Regressione polinomiale: particolare tipologia in cui si trasformano i dati iniziali in dati polinomiali di un certo grado a cui successivamente si applica un modello di regressione lineare.
  Si consideri ad esempio la funzione descritta nel modello di regressione lineare semplice, ovvero $Y = a + bX$, che viene perciò trasformata in:
  $$Y = a + bX + cX^2$$
  È da notarsi che si tratta ancora di un modello lineare, ma ora la funzione viene rappresentata da una curva piuttosto che da una retta:

[2]Si rimanda il lettore al capitolo 5, dove verrà spiegato con maggiore chiarezza la natura di tale stimatore, utilizzato infatti come uno tra gli indici di qualità dei risultati.

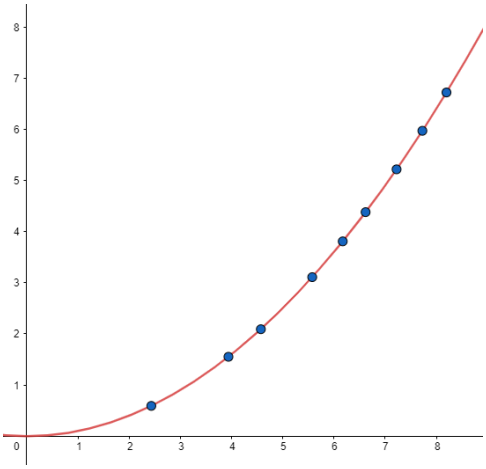


Figura 2.4: Modello di regressione polinomiale overfitted

In particolare, più la curva è rappresentata da una funzione di grado maggiore, più essa soffrirà del fenomeno di overfitting, imparando perciò anche il rumore nei dati;

- Regressione con Support Vector: il concetto di SVR consiste nell'identificare un iperpiano con un determinato margine in cui sono contenuti il maggior numero di punti rappresentanti i dati.
  Perciò, invece di minimizzare la funzione di costo e il conseguente rapporto tra valore attuale e predetto, con questa tecnica si cerca di adattare l'errore entro certi confini, quelli dell'iperpiano appunto:

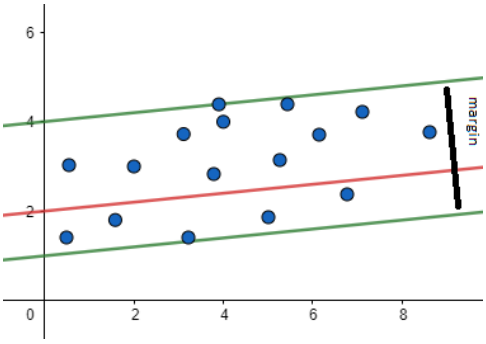


Figura 2.5: Modello di regressione con Support Vector

- Regressione con alberi decisionali: questi ultimi possono essere utilizzati sia in classificazione che in regressione e in questo secondo caso, ci sono vari algoritmi che funzionano egregiamente e per la

maggior parte lo fanno attraverso la riduzione della deviazione standard (l'information gain invece è usato nella classificazione).
Più precisamente, un albero decisionale è costruito partizionando i dati in sottoinsiemi contenenti istanze con valori simili e proprio la deviazione standard è usata per calcolare l'omogeneità di un insieme di dati numerici.

- Regressione con Random Forest: questo genere di regressori fa parte dei cosiddetti metodi di insieme, ovvero particolari tecniche che vengono sviluppate partendo da metodologie di regressione (o classificazione) più semplici[24].
  In breve, il Random Forest opera come segue:

  1. Vengono selezionati K punti (ovvero K dati iniziali);
  2. Si identifica un numero n che consiste nel numero di regressori ad albero decisionale da creare e poi vengono effettivamente istanziati;
  3. Il valore medio di ogni ramo dell'albero è assegnato ad un nodo foglia in ogni albero decisionale;
  4. Per predire un output, viene preso in considerazione il valore medio di tutte le predizioni di tutti gli alberi decisionali.

  Questa particolare tecnica previene il fenomeno dell'overfitting, molto comune negli alberi decisionali, creando piccoli sottoinsiemi randomici di attributi dei dati iniziali e istanziando alberi decisionali più piccoli a partire da questi nuovi dati.

2.2 Reti neurali

Le reti neurali artificiali sono modelli matematici composti da neuroni artificiali che si ispirano al funzionamento biologico del cervello umano e che sono utilizzati al giorno d'oggi nell'ambito del machine learning e dell'intelligenza artificiale.
In particolare, in questo sottocapitolo verranno analizzate le caratteristiche generali di questi modelli e ne verranno descritte le tipologie effettivamente utilizzate nell'elaborato di tesi.
Occorre inizialmente però illustrarne la struttura: le reti neurali sono composte principalmente da un aggregato di nodi chiamati neuroni, particolari elementi che hanno la capacità di prendere in input dei dati, elaborarli e restituire un risultato in output:

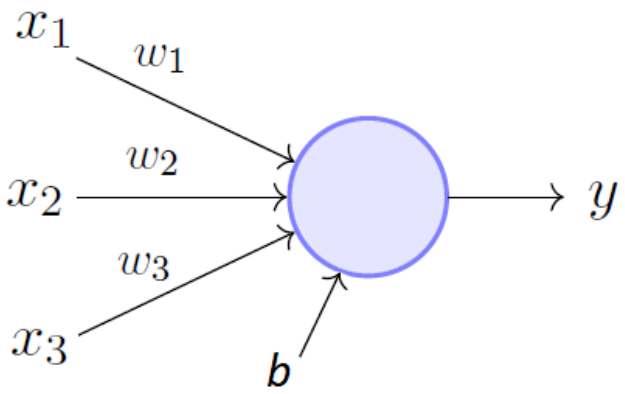


Figura 2.6: Neurone artificiale

Più formalmente, quello che succede attraverso un neurone artificiale, è descritto come segue:

1. Ogni input viene moltiplicato per il corrispettivo peso:

$$x_1 \rightarrow x_1 * w_1$$

$$x_2 \rightarrow x_2 * w_2$$

$$x_3 \rightarrow x_3 * w_3$$

2. I nuovi input vengono sommati assieme ad un altro particolare input, chiamato bias (b):

$$(x_1 * w_1) + (x_2 * w_2) + b$$

3. Infine, questa somma è passata ad una particolare funzione, chiamata funzione di attivazione:

$$y = f((x_1 * w_1) + (x_2 * w_2) + b)$$

   In particolare, questa funzione viene scelta a seconda dell'obiettivo del modello, poiché diverse funzioni possono dare risultati e performance nettamente differenti[25].
   Infatti, tali funzioni determinano l'output del neurone e più in generale del modello di rete neurale, l'accuratezza finale e anche l'efficienza computazionale dell'allenamento del modello stesso.
   Inoltre, esse hanno anche un forte impatto sull'abilità della rete di convergere e sulla velocità nel farlo (ad esempio, alcune funzioni

non adatte alla risoluzione di un particolare problema in analisi ma utilizzate comunque, potrebbero proprio non fare convergere).
Intuitivamente, esse determinano se il neurone deve essere attivato o meno e quindi l'idea base per implementarla potrebbe essere quella di codificarle come un gate logico che restituisce 0 o 1 sulla base di una soglia, utilizzando perciò una funzione a step:

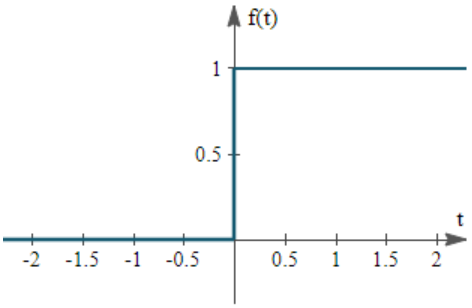


Figura 2.7: Step activation function

Tale approccio funziona solamente negli scenari di classificazione dove occorre appunto dare un'etichetta precisa in output e quindi restituire 0 o 1.
In tutti gli altri casi in cui si voglia piuttosto restituire la probabilità in cui l'esempio da classificare appartenga ad una certa categoria, si utilizzano quindi altre funzioni di attivazione.
Tra queste in particolare, le più utilizzate sono la sigmoide, la ReLu (Rectified Linear Unit) e la softmax, funzioni non lineari che creano complessi mapping tra gli input e gli output della rete e che sono necessarie per imparare e modellare dati in forma complessa come immagini, video, audio o semplicemente dataset che non sono lineari o che hanno una grande dimensionalità.

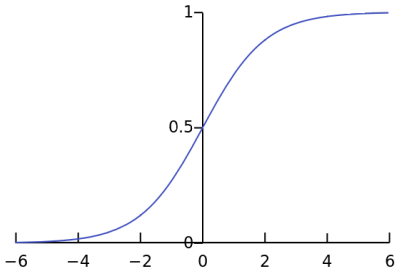


Figura 2.8: Sigmoid activation function

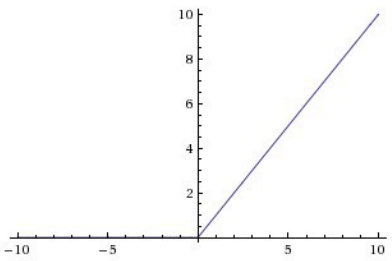


Figura 2.9: ReLu activation function

Una rete neurale è perciò un insieme di neuroni connessi tra loro in una delle infinite possibili topologie che prevede la loro disposizione su tre tipologie di layer diversi:

- Input layer: layer a cui vengono connessi gli input del modello. In particolare, esiste un nodo per ogni attributo nel dato del training set;
- Hidden layer: layer intermedio tra quello di input e quello di output. Il numero di nodi può essere di cruciale importanza per lavorare in domini applicativi molto complessi, tuttavia può portare all'incremento del fenomeno dell'overfitting;
- Output layer: layer di output del modello, il cui numero di neuroni è legato essenzialmente al numero totale possibile di classi od etichette a cui associare un dato in input.

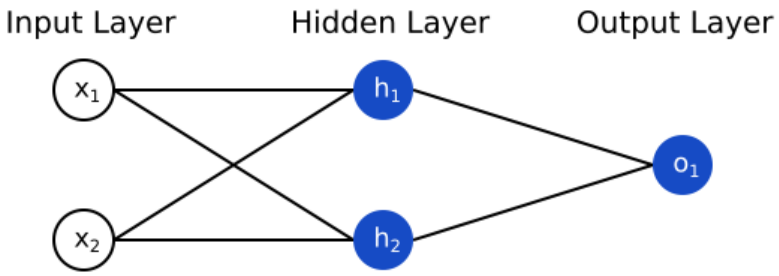


Figura 2.10: Modello di rete neurale

L'algoritmo per l'addestramento di una rete neurale consta essenzialmente di due fasi: nella prima, detta di feedforward, i neuroni vengono alimentati dagli esempi del training set, e successivamente, procedendo in avanti, la rete andrà a calcolare l'output di ogni neurone in ciascun livello.
La metodologia utilizzata è simile a quella descritta nel sottocapitolo precedente per quanto riguarda i regressori, in quanto anche qui occorre stimare la bontà dell'output, ovvero quanto il valore predetto si discosti effettivamente da quello reale.
Successivamente, nella fase di backpropagation, si vanno a misurare i contributi all'errore di ciascun neurone fino allo strato di input.
Tutto ciò viene eseguito in quanto l'obbiettivo è appunto la modifica del valore dei pesi per ridurre l'errore, approccio portato a termine anche qui con il metodo della discesa del gradiente.

Infatti, per scegliere come modificare i pesi, si segue la direzione opposta indicata dal gradiente: esso indica infatti la direzione di crescita di una funzione per cui, seguirne la direzione opposta, permette di minimizzare la funzione stessa.
In definitiva, anche qui perciò allenare il modello di rete neurale significa in un certo senso andare a diminuire via via la loss (utilizzando ad esempio MSE come indicatore) fino a raggiungere un errore minimo o comunque accettabile.

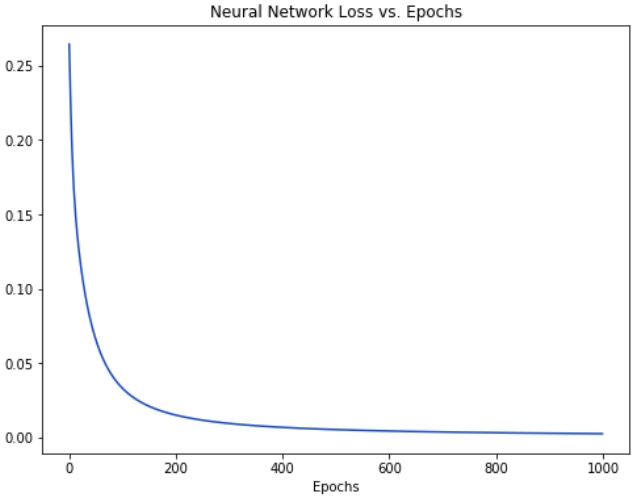


Figura 2.11: Esempio di training di rete neurale: loss in decrescita all'aumentare delle epoche

Sulla base di queste conoscenze comuni, configurando i vari iperparametri della rete e modificandone la topologia a seconda dei casi d'uso, sono stati sviluppati poi in letteratura diversi modelli di rete neurale, ognuna appunto più efficace in uno specifico dominio applicativo.
In particolare, verranno utilizzati nell'elaborato due modelli specifici di rete neurale: le deep neural networks e le graph convolutional networks.

2.2.1 Deep neural networks

Le deep neural networks rappresentano di fatto una classe di algoritmi di machine learning che utilizza una topologia più complessa organizzata a più livelli per estrarre ed apprendere progressivamente dati sempre più ad alto livello partendo dal semplice input dato in pasto alla rete.
Ad esempio, nel processing delle immagini, si utilizzano le reti neurali

convoluzionali in cui i livelli più vicini all'input sono incaricati di identificare i vertici, gli spigoli e più in generale i confini dell'immagine, mentre quelli più ad alto livello si occupano di ricercare concetti più rilevanti per l'essere umano, come il riconoscimento di caratteri, di numeri o di volti. Una rete neurale di questo tipo è perciò simile ad un modello semplice, ma in più è caratterizzata da un maggior numero di hidden layer, che svolgono appunto estrazioni ed operazioni intermedie sui dati.
Inoltre, si parla di layer densi, in quanto tutti i neuroni di un layer sono collegati a tutti quelli del layer successivo.

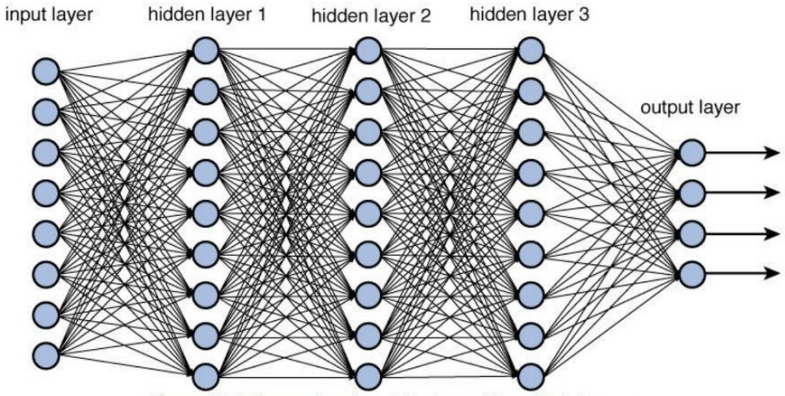


Figura 2.12: Esempio di architettura deep di una rete neurale

La configurazione del numero di nodi per ogni hidden layer e il numero stesso di hidden layer è un argomento non affatto banale e rappresenta perciò un ulteriore livello di difficoltà nel creare un nuovo modello di rete neurale profonda.
A tal proposito, esiste un teorema di approssimazione universale che stabilisce che una rete neurale feed-forward, con un singolo hiddel layer costituito da un numero finito di neuroni, può approssimare funzioni continue facendo leggere assunzioni sulla funzione di attivazione[26][27]. Questa teoria, sebbene specifichi che in pratica occorre una rete neurale ad un solo hiddel layer per risolvere qualsiasi problema, non specifica però quanto tempo effettivo essa impieghi per imparare qualcosa.
Perciò, il tuning dei layer e dei nodi relativi rappresenta un problema ulteriore al fine della modellazione, e come si vedrà successivamente, verranno proposti in questo elaborato più modelli a diversi layer e numero di nodi, al fine di trovare il giusto tradeoff tra complessità e precisione della rete.

2.2.2 Graph convolutional networks

Molti dataset reali contengono informazioni e quindi dati organizzati in grafi: è il caso ad esempio di quelli derivanti dai social network, dal World Wide Web o addirittura dalla reti create per l'interazione delle proteine. Negli ultimi anni, molti approcci sono stati sviluppati per generalizzare il modo in cui lavorare su grafi strutturati in maniera arbitraria[28][29][30][31].
Sicuramente però, l'approccio più riconosciuto al giorno d'oggi è quello dato dal lavoro di Kipf e Welling del 2017[13], in cui viene trattato l'argomento delle graph gonvolutional networks (GCN), tipologie di reti neurali convoluzionali pensate appunto per lavorare su strutture a grafo. Per questi modelli, l'obiettivo è quello di imparare una funzione su un grafo $G = (V, \epsilon)$ che prende in input:

- X, ovvero una matrice di features $N \times D$, dove N è il numero di nodi del grafo e D è il numero di features in input per ogni nodo;
- A, ovvero una matrice $N \times N$ che rappresentata la struttura del grafo e che comunemente coincide con la matrice di adiacenza. Quest'ultima è una matrice binaria quadrata che ha come indici di righe e colonne i nomi dei vertici del grafo: nel posto (i, j) della matrice si trova un 1 se e solo se esiste nel grafo un arco che va dal vertice i al vertice j, altrimenti si trova uno 0.

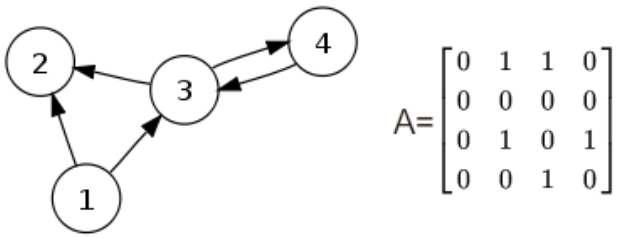


Figura 2.13: Esempio di matrice di adiacenza

Tale funzione produce un output $Z = N \times F$, dove F è il numero delle features in output in ogni nodo.
Inoltre, ogni layer della rete neurale può essere scritto come una funzione non lineare

$$H^{l+1} = f(H^{(l)}, A)$$

con $H^{(0)} = X$ e $H^{(L)} = Z$, dove L è il numero di layer. In pratica, ad ogni layer corrisponde una matrice di features $N \times F$ in cui ogni riga

rappresenta le features di un nodo appunto.
Ad ogni layer infatti, esse sono aggregate per formare quelle del prossimo layer, usando appunto la funzione f. In questo modo, le features diventano più astratte in maniera incrementale ad ogni layer consecutivo.
Di conseguenza, specifici modelli differiscono solo nel modo in cui $f(\cdot, \cdot)$ è scelta e parametrizzata.
Più in dettaglio, si consideri ad esempio la seguente funzione di propagazione:

$$f(H^{(l)}, A) = \sigma(AH^{(l)}W^{(l)})$$

con $W^{(l)}$ intesa come la matrice dei pesi per il layer l-esimo e $\sigma(\cdot)$ una funzione di attivazione non lineare come ReLU.
Le GCN presentano due limitazioni iniziali, facilmente risolvibili:

1. Moltiplicare per A, significa che, per ogni nodo, si sommano tutti i vettori di features di tutti i nodi vicini ma non quelle del nodo stesso (a meno che ci siano self-loops nel grafo). Si risolve questo problema forzando i self-loops nel grafo, se non sono presenti, e quindi in pratica si aggiunge la matrice identità ad A.

2. A tipicamente non è normalizzata e quindi moltiplicare per tale matrice cambia totalmente la scala dei vettori di features.
   Per risolvere ciò, si fa in modo che tutte le righe di A sommate diano come risultato 1, ovvero effettuando l'operazione $D^{-1}A$, dove D è la matrice di grado, cioè la matrice diagonale che contiene le informazioni sul grado di ogni vertice del grafo, cioè il numero di archi che sono collegati ad esso.
   In particolare, considerato un grafo G, la matrice di grado D per G è una matrice diagonale $N \times N$ tale che:

$$d_{i,j} := \begin{cases} deg(v_i) & se\ i = j \\ 0 & altrimenti \end{cases}$$

   dove il grado $deg(v_i)$ di un vertice è il numero di archi che termina in un dato vertice.
   In un grafo non orientato ciò significa che ogni nuovo ciclo aumenta il grado di un vertice di due. In un grafo orientato, invece, il termine grado può riferirsi al numero di archi in entrata o al numero di archi in uscita di un tale vertice.
   Moltiplicare per $D^{-1}A$ corrisponde essenzialmente a prendere il valore medio delle features dei nodi vicini ma nella pratica, si ottiene un modello migliore quando si utilizza invece una normalizzazione simmetrica, cioè $D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$.

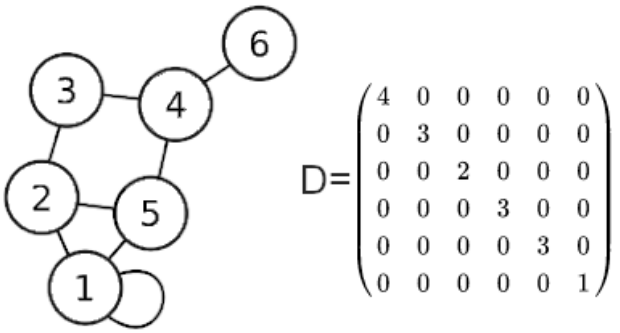


Figura 2.14: Esempio di matrice di grado

Combinando le due risoluzioni appena descritte si ottiene la funzione di propagazione essenzialmente descritta e introdotta da Kipf e Welling[13]:

$$f(H^{(l)}, A) = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

con $\hat{A} = A + I$, dove I è la matrice identità e $\hat{D}$ la matrice di grado di $\hat{A}$.

Il lavoro appena descritto ad opera di Kipf e Welling è poi stato discusso ed infine implementato nei diversi strumenti ad oggi disponibili per la creazione ed il training delle reti neurali.
Tra questi troviamo il framework ad opera di Grattarola, Spektral[32], ad oggi uno degli strumenti sicuramente più completi che sfruttano le GCN e l'integrazione con Keras, framework utilizzato in larga scala in ambito machine learning.

3 Strumenti utilizzati

In questo capitolo verranno analizzati e descritti in breve tutti gli strumenti tecnici e i framework utilizzati nell'elaborato, focalizzandosi soprattutto sulle funzionalità proposte effettivamente adoperate.
In sostanza, verrà discussa la motivazione della scelta di Tensorflow, piattaforma di Google per il machine learning, come backend di supporto al framework Keras.
Inoltre, si affronteranno tematiche inerenti a ScikitLearn, tool che offre molti strumenti Python sempre in ambito machine learning, fino ad arrivare a Spektral, framework maggiormente utilizzato in ambito GCN.
Infine, verrà fatto un breve excursus sull'ambiente di test utilizzato nell'elaborato, D.A.V.I.D.E., concentrandosi in particolare sulla sua architettura e sulla sua maniera proposta con cui potersi interfacciare.
Proprio in merito a questo, poi si parlerà di Slurm, celebre programmatore di lavoro opensource su piattaforme Unix.

3.1 Keras

Al giorno d'oggi, lo studio delle tecniche di machine learning e più in generale di intelligenza artificiale, non è più qualcosa di solamente teorico, ma è diventato parte applicativa integrante dei modelli tecnologici più moderni ed evoluti.
In questo senso, nel tempo sono stati sviluppati strumenti efficaci nell'ambito machine learning, in grado di nascondere la complessità teorica dei modelli e di fornire metodi e API più ad alto livello in maniera da essere più comprensibili e facilmente utilizzabili dagli sviluppatori.
Ci sono state perciò parecchie proposte di framework, e tra queste si trova appunto Keras, uno degli strumenti principali più utilizzato ad oggi nello sviluppo di modelli di machine learning.
Infatti, Keras è stato creato con lo scopo di essere user-friendly, modulare, facile da utilizzare e in grado di lavorare con Python.
Perciò, ad esempio i layer delle reti neurali, le funzioni di costo, gli ottimizzatori e le funzioni di attivazione sono tutti moduli standard in Keras che possono essere combinati facilmente al fine di creare nuovi modelli.
Il motivo dell'ascesa della popolarità di Keras deriva anche dalla facilità di adozione in scenari molto diversi, dal supporto di un grandissimo numero di opzioni di deployment e dalla forte compatibilità con modelli hardware costituiti da molte GPUs per il training distribuito.
In più, Keras è sostenuto da Google, Microsoft, Amazon, Apple, Nvidia, Uber e altri colossi del mondo tecnologico e non.

Figura 3.1: Logo di Keras

Tra i moduli principali del suddetto framework troviamo sicuramente il cosiddetto Model, la struttura dati primaria nel core di Keras.
Tramite esso, è facilmente possibile creare reti neurali in maniera molto veloce e con l'utilizzo di poche API ad alto livello: ad esempio, scegliendo un modello di tipo Sequential, ovvero una semplice aggregazione di layers, inizialmente sarà sufficiente solamente deciderne la topologia aggiungendo livelli e decidendo come essi debbano essere legati tra di loro.
Sarà poi da definire una loss function da minimizzare, un compilatore per effettuare la discesa del gradiente ed invocare la funzione fit, che avvierà di fatto il training vero e proprio del modello.

Come si diceva precedentemente, una serie molto ampia di opzioni e API funzionali è disponibile al fine di controllare ancora più finemente tutti i passi dell'addestramento della rete e della corretta terminazione di essa. Inoltre, sono disponibili moltissimi dataset standard facilmente importabili nel proprio modello, feature senz'altro fondamentale per gli sviluppatori alle prime armi.
Altro concetto importante è quello del backend: Keras infatti di per sè offre un framework ad alto livello per la costruzione di modelli di machine learning, nascondendo perciò la complessità sottostante.
Tale layer a basso livello è infatti il cosiddetto backend, ovvero tutte quelle operazioni nascoste allo sviluppatore necessarie per la creazione di un modello tra cui si trovano sicuramente tutte quelle tra i tensori, come prodotti e convoluzioni.
Per essere il più indipendente possibile dalle librerie sottostanti, la scelta del backend non è univoca, ma sono disponibili diverse possibilità:

- Tensorflow: framework opensource simbolico per la manipolazione di tensori sviluppato da Google;
- Theano: framework opensource simbolico per la manipolazione di tensori sviluppato dal LISA Lab dell'Université de Montréal;
- CNTK: toolkit opensource per il deep learning sviluppato da Microsoft.

Nel seguente elaborato in particolare è stato adottato Tensorflow come backend per una serie di motivazioni:

1. Esiste una community di sviluppatori dietro al progetto che permette il reperimento in maniera facile di informazioni da implementare nel proprio modello in sviluppo.
   Inoltre è possibile risolvere molto velocemente eventuali problemi emersi durante lo sviluppo, grazie alla consultazione di tantissimi forum online in cui reperire sviluppatori che hanno affrontato difficoltà simili;
2. Non ci sono i cosiddetti vendor lock-in, ovvero non ci sono vincoli se in futuro si sceglie di cambiare backend, in quanto Tensorflow è totalmente opensource;
3. Si sta spostando velocemente dall'ambiente sperimentale a quello di produzione, in quanto sta venendo largamente utilizzato da grandi colossi in tutti i settori che ne stanno affinando la compatibilità con i diversi ambienti e testando l'affidabilità;

4. L'adozione è in crescita esponenziale e molti libri sono stati e stanno venendo scritti in maniera da avvicinare a questo backend molti altri sviluppatori e risolvere tutti gli eventuali dubbi emersi;

5. Si tratta di un backend completamente estensibile e facilmente integrabile con molte strumentazioni e codice ad-hoc.

Keras è stato utilizzato nella tesi principalmente come framework di supporto per la creazione di modelli di reti neurali.

3.2 Tensorflow

Tensorflow è una libreria opensource inizialmente sviluppata dal Google Brain Team, parte del gruppo di ricerca per il machine learning ed il deep learning.
Esso utilizza Python per fornire API lato frontend mentre a basso livello esegue tali applicazioni utilizzando C++ ad alte performance, permettendo agli sviluppatori di creare i cosiddetti dataflow graphs, ovvero strutture che descrivono come i dati si muovono attraverso un grafo o una serie di nodi che svolgono certe operazioni.

Figura 3.2: Logo di Tensorflow

Tutto questo viene fornito appunto attraverso l'utilizzo di Python, che riesce ad esprimere al meglio come astrazioni ad alto livello possano interagire tra di loro.
Come si diceva però, tutte le operazioni matematiche sono effettuate tramite codice binario in C++, in quanto Python si limita solamente a dirigere là la computazione fornendo strumenti però più ad alto livello.
La base di calcolo di Tensorflow può essere pressochè eseguita su qualsiasi dispositivo, tra cui la propria macchina in locale, un cluster nel cloud, i dispositivi sia Android che Apple, le CPUs e le GPUs, con performance logicamente diverse.

Esiste addirittura un'ottimizzazione del framework se si decide di eseguirlo sul cloud di Google, che permette prestazioni ancora migliori.
Inoltre, supportando la parallelizzazione, si possono allenare complessi modelli con training set giganteschi in tempi relativamente brevi dividendo i calcoli su cluster di macchine.
Tensorflow 2.0, distribuito dall'ottobre del 2019, ha rinnovato il framework in molti modi, basandosi principalmente sul feedback degli utenti, per fare in modo di poterci lavorare più facilmente e in maniera più performante.
L'allenamento distribuito è ancora più facile grazie alle nuove API, ed è stato introdotto anche Tensorflow Lite, che rende possibile il deployment dei modelli in un range di scenari ancora più vasto.
Sfortunatamente, una gran parte di codice scritto per versioni precedenti di Tensorflow non è del tutto compatibile con la nuova versione del framework, e deve perciò essere riscritto per sfruttarne i principali nuovi vantaggi.
Per questo motivo, nell'elaborato di tesi si è deciso di utilizzare Tensorflow come backend nella sua versione appena precedente alla 2.0, in maniera da sfruttare API ampiamente testate e non dover incorrere in eventuali problemi di compatibilità.

3.3 ScikitLearn

Scikit-learn è una libreria open source per il machine learning in ambienti di programmazione Python.
Contiene algoritmi di classificazione, regressione e clustering e macchine a vettori di supporto, regressione logistica, classificatori bayesiani, k-mean e DBSCAN, ed è progettato per operare con le librerie NumPy e SciPy.

Figura 3.3: Logo di Scikit-learn

In particolare fornisce:

- Tool semplici ed efficaci per l'estrazione e l'analisi dei dati;
- Un grande supporto da parte della community di sviluppatori;
- Estensibilità con codice e tools ad-hoc.

Scikit-learn è stato utilizzato nella tesi principalmente come framework di supporto per la creazione di modelli di regressione lineare.

3.4 Spektral

Spektral è una libreria per il deep learning sui grafi, basata su Keras e Tensorflow, sviluppata da Daniele Grattarola[32].
L'obbiettivo principale del progetto è quello di fornire un framework semplice ma completo per creare e modellare le graph neural networks.
Implementa perciò un gran numero di layer per il learning sui grafi, tra cui appunto le graph convolutional networks (GCN), le Chebyshev networks (ChebNets), le graph attention networks (GAT) e le graph isomorphism networks (GIN).

Figura 3.4: Logo di Spektral

Più specificamente il framework è stato scelto poiché implementa la prima tipologia, le reti neurali effettivamente utilizzate nell'elaborato di tesi, basandosi in toto sul lavoro di Kipf e Welling, già discusso nel sottocapitolo 2.2.2[13].
In particolare, Spektral è creato come estensione delle API di Keras, in maniera da renderne l'implementazione estremamente semplice per i principianti, mantenendo tuttavia flessibilità per i più esperti del dominio e per i ricercatori.
Più in dettaglio quindi, per implementare una GCN occorrerà definire nel modello anche layer nuovi, i cosiddetti GraphConv layers, che modellano la logica della rete convoluzionale che agisce sui grafi. In particolare, come si vedrà nel capitolo inerente al progetto vero e proprio, occorrerà

definire il modello attraverso una serie di tensori organizzati in maniera consecutiva. Il resto delle API è sostanzialmente invariato rispetto a quelle proposte da Keras.
Un aspetto fondamentale del framework è la maniera con cui esso si può configurare per la creazione di reti neurali che lavorano sui grafi: Spektral utilizza rappresentazioni matriciali dei grafi, in perfetto allineamento con le modalità teoriche presentate da Kipf e Welling. Tale approccio infatti è quello migliore finora sperimentato ed è perfetto per fare della computazione parallela sulle GPU.
In particolare, il framework si aspetta come parametri di ingresso esattamente gli stessi previsti da Kipf e Welling e descritti nel sottocapitolo 2.2.2 del presente elaborato di tesi. In più però, alcune funzionalità sono implementate per lavorare su un singolo grafo, mentre altre considerano anche batch di grafi. A tal proposito ad esempio, il dataset CORA che rappresenta una fitta rete di citazioni tra vari articoli[33], funziona con la prima modalità di utilizzo in quanto si è interessati nei nodi individuali e nelle connessioni tra di essi, che sono specifici per ogni rete neurale e in cui i nodi stessi rappresentano proprio i dati del modello.
La seconda modalità di funzionamente invece, può essere spiegata andando a considerare come esempio la classificazione delle proprietà chimiche delle molecole in cui ogni molecola è un esempio nel dataset, e gli atomi e i legami tra di essi sono parte costituente del dato in sè.
Quindi, in questo scenario si è interessati a trovare dei pattern che descrivono le proprietà delle molecole in generale e per questo motivo, il secondo approccio è quello consigliato.
In realtà comunque le modalità di funzionamento sono tre:

1. singola: dove si ha un singolo grafo, con una topologia e degli attributi prefissati;

2. a batch: dove esistono più grafi, ognuno con la propria topologia e i propri attributi;

3. mista: dove si ha un grafo con una topologia predefinita ma un insieme di differenti attributi.

Definite come A la matrice di adiacenza, N il numero dei nodi, X la matrice degli attributi del nodo, F la dimensione degli attributi del nodo, E la matrice degli archi degli attributi ed S la dimensione di quest'ultima matrice, le differenze precise tra le varie modalità sono rappresentate dalla figura sottostante:

Mode	A.shape	X.shape	E.shape
Single	(N, N)	(N, F)	(N, N, S)
Batch	(batch, N, N)	(batch, N, F)	(batch, N, N, S)
Mixed	(N, N)	(batch, N, F)	(batch, N, N, S)

Figura 3.5: Differenze tra le varie modalità di funzionamento

In particolare poi, il menzionato GraphConv è l'implementazione pratica di quanto scoperto da Kipf e Welling. Esso può lavorare in una delle tre modalità appena descritte e fondamentalmente effettua la computazione

$$Z = \sigma(\tilde{A}XW + b)$$

dove X è la matrice delle features del nodo, $\tilde{A}$ è la laplaciana normalizzata, W è il kernel convoluzionale, b è un vettore di bias, e σ è la funzione di attivazione.
Spektral è stato utilizzato nella tesi principalmente come framework di supporto per la creazione di modelli di GCN, in modalità mista[1].

3.5 Slurm

Slurm (Simple Linux Utility for resource Management) è un programmatore di lavoro, o workload manager, opensource su piattaforme Unix, utilizzato da molti dei supercalcolatori diffusi globalmente.
Fornisce in particolare tre funzionalità chiave:

1. permette di allocare accessi esclusivi o non esclusivi alle risorse (ovvero ai nodi del computer) da parte degli utenti ma per un periodo di tempo limitato;

2. fornisce un framework completo per inizializzare, eseguire e monitorare processi (tipicamente MPI) su un set di nodi;

3. offre complessi sistemi di esecuzione parallela di job che si appoggiano su code.

[1]Le motivazioni del perchè di tale scelta sono brevemente riportate nella parte del progetto, precisamente nel capitolo 4.5.2.

Figura 3.6: Logo di Slurm

Inoltre è provvisto di molte altre funzionalità che lo rendono un sistema molto affidabile e di fatto installato sul 60% dei supercalcolatori disponibili globalmente. Tra queste occorre menzionare sicuramente l'architettura a nodi che non prevede un single point of failure, una scalabilità altissima (si possono schedulare fino a 100000 job indipendenti), delle performance incredibili, una grande possibilità di configurazione di ogni aspetto e una gestione precisissima di utenti e della sicurezza tramite l'adozione di ruoli applicativi.
Tale manager è quello utilizzato tutt'ora dal supercomputer adoperato per il testing dei modelli di deep learning descritti nel capitolo successivo di questo elaborato, l'unico in grado di calcolare e restituire dei risultati in maniera veloce, considerando le grandi capacità di elaborazione e di computazione richieste.
D.A.V.I.D.E. (Development for an Added Value Infrastructure Designed in Europe) è un supercalcolatore progettato da E4 Computer Engineering S.p.a per PRAC (Partnership for Advanced Computing in Europe). Esso è composto di fatto da 45 nodi connessi tra loro su un'architettura OpenPower NViDIA NVLink e processori OpenPower8 NVIDIA Tesla P100 SXM2.
Gli script di progettazione utilizzati per avere dei risultati tangibili in questo elaborato sono stati adattati per essere compatibili con l'architettura di D.A.V.I.D.E. ed in particolare con il suo workload manager.
Per la descrizione esaustiva delle tecniche adoperate per fare ciò, si rimanda il lettore alla parte progettuale della tesi, specificamente al sottocapitolo 4.4.

3.6 Altre librerie

A completare la panoramica degli strumenti utilizzati, è importante anche indicare quali sono state le principale librerie di supporto nella stesura del codice Python.
Tra queste, le principali sono:

- Pandas: libreria che fornisce strutture dati ad alte performance e tool di analisi particolarmente efficaci;
- Matplotlib: libreria per tracciare grafici 2D, utilizzando apposite API per una grandissima varietà di elementi;
- PyQt5: libreria che rende possibile l'utilizzo di Qt, framework C++ per la creazione di GUI, all'interno dell'ambiente Python.

Figura 3.7: Loghi delle librerie sopramenzionate

4 Progetto

In questo capitolo si affronterà in dettaglio tutto ciò che riguarda lo sviluppo del progetto e le tecniche correlate adoperate.
Partendo dai dataset generati automaticamente sulla base di script ad-hoc, si testeranno prima modelli semplici come i regressori lineari, fino ad arrivare a modelli complessi di deep learning, passando per una modellazione esaustiva di molte tipologie di reti neurali a diversi layer.
Infine, l'introduzione di vincoli dati dal grafo darà modo di testare ulteriori modelli di machine learning come ad esempio le GCN, ovvero le graph convolutional networks.

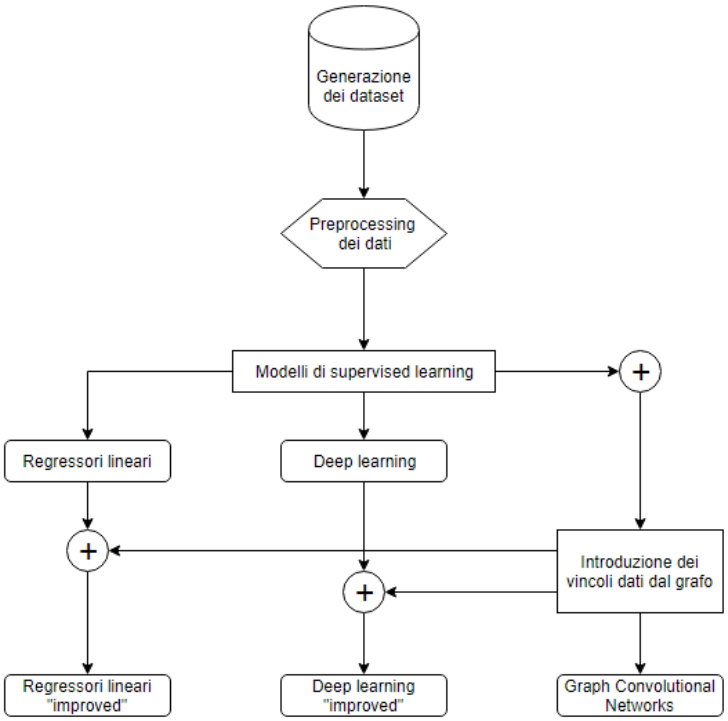


Figura 4.1: Workflow del progetto

4.1 Creazione dei dataset iniziali

La generazione dei dataset è il primo passo per ottenere dei modelli su cui lavorare: in particolare essi costituiscono proprio il punto di partenza del progetto e di fatto sono coloro che poi andranno a generare il training set che servirà nel testing dei diversi modelli di machine learning.
Come si diceva all'inizio, vengono considerati nel progetto solamente i benchmark più significativi, ovvero Convolution, Correlation, DWT, BlackScholes e Jacobi e proprio con il loro utilizzo vengono generati i dataset iniziali, rappresentati come file .csv.
Al loro interno sostanzialmente si trovano delle tabelle: per ogni riga ci sono n colonne che descrivono una precisa configurazione (dove n dipende dal numero delle variabili dello specifico benchmark) e m colonne che descrivono l'errore (m dipende dal numero degli input set considerati, 30 in questo progetto).
In particolare, l'errore in ognuna delle m colonne è la differenza tra il risultato del benchmark con le variabili impostate alla massima precisione e il risultato del benchmark con le variabili alla precisione stabilita dalla specifica configurazione del dataset (come si diceva anche nel sottocapitolo 1.2).

4.1.1 Operazioni di campionamento

Il primo passo consiste essenzialmente nel scegliere le configurazioni di testing, ovvero il numero di bit (da 3 a 53) da assegnare ad ogni variabile, con le quali poi calcolare gli errori utilizzando gli specifici benchmark.
Le configurazioni sperimentali sono perciò ottenute grazie all'LHS (Latin Hypercube sampling), un metodo statistico per generare campioni semicasuali di valori da una distribuzione multidimensionale.
Questo tipo di campionamento permette di ottenere uniformemente dei valori e ha lo scopo di spargere i punti di campionamento più precisamente su tutti i possibili valori invece che affidarsi alla completa casualità.
Considerando un campionamento su due dimensioni, con l'LHS occorre prima decidere quanti punti utilizzare e per ognuno di essi, ricordarsi in quale riga e colonna essi sono stati scelti.
Questo approccio è molto diverso da un approccio più casuale in cui i nuovi punti sono generati senza tenere in considerazione quelli creati precedentemente e dove non occorre nemmeno sapere in anticipo quanti punti di campionamento sono necessari.
Inoltre l'LHS è utilizzato come tecnica nel campionamento ortogonale, dove lo spazio viene diviso in sottospazi ugualmente probabili e tutti i

punti sono scelti allo stesso momento, assicurandosi che l'insieme complessivo sia un LHS e che ogni sottospazio sia campionato con la stessa densità.

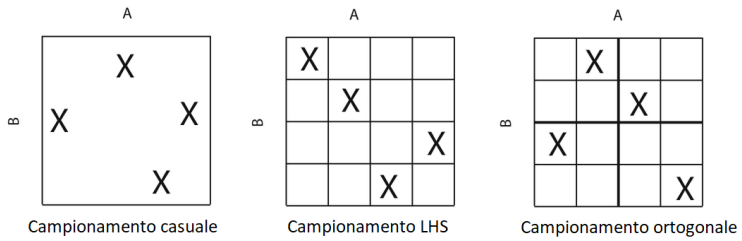


Figura 4.2: Tecniche di campionamento a confronto

4.1.2 Generazione dei dati

Un algoritmo di generazione di dataset è eseguito per creare nuove strutture dati con cui lavorare e più precisamente ogni dataset risultante è inerente ad uno specifico benchmark e a delle specifiche configurazioni precedentemente campionate.
Inoltre, per ogni singola configurazione vengono utilizzati più input set per generare errori. Si tratta di un approccio nuovo, non esplorato finora, che permette di lavorare su più dati per creare modelli più ricchi nelle conseguenti metodologie di machine learning.
Il reale funzionamento dell'algoritmo Python prevede che, dopo aver stabilito quali sono le cartelle di destinazione e gli input set su cui lavorare, esso inizi la vera e propria computazione considerando una configurazione alla volta e un input set alla volta.
In particolare, per effettuare i calcoli dell'errore, sono utilizzati dei cosiddetti target files che contengono i risultati alla massima precisione.
Infine, applicando la formula descritta nel sottocapitolo 1.2, si ottengono effettivamente gli errori inerenti ai rispettivi input set.
Per fornire ulteriori informazioni e dati su cui lavorare, vengono inoltre generati dei dataset inerenti ai cosiddetti vicini: essi si riferiscono a delle configurazioni vicine in cui ogni variabile è scelta nella distanza euclidea rispetto a quella corrispondente nella configurazione iniziale.
Infine, anche tramite queste configurazioni un nuovo dataset viene generato utilizzando le stesse regole per calcolare gli errori.

4.1.3 Approccio analitico

Prima di procedere alla creazione di modelli di machine learning, i dataset devono essere analizzati in maniera da estrapolare eventualmente informazioni utili sulle relazioni tra una configurazione e gli errori specifici per ogni input set considerato.
Per farò ciò sono stati creati degli script che sostanzialmente leggono i dataset in diverse dimensioni:

- Orizzontalmente: lo script analizza un dataset relativo ad uno specifico benchmark estraendo n configurazioni casuali specificate come argomento e fornendo dei grafici per comprenderne meglio i dati. Le configurazioni sono estratte in realtà non in maniera completamente casuale: prendendone n dal totale non avrebbe molto senso poiché sarebbe molto più significativo sceglierne dei campioni rappresentativi per fare un'analisi più accurata.
  Per fare ciò, tutte le configurazioni sono prima lette e ordinate in base al numero totale di bit. Successivamente, il vero campionamento casuale è avviato su questa lista ordinata: per esempio, se le configurazioni da ricercare e analizzare sono 10, l'insieme totale di esse viene prima ordinato e poi diviso in 10 sottoinsiemi, in cui viene poi preso il campione in maniera casuale.
  Lo script poi prende come parametro il nome del dataset e un bit, che può valere 1 o 0. Quest'ultimo permette di fare due distinte tipologie di analisi, una sull'intero dataset e l'altra su una sua versione filtrata.
  Questa seconda modalità in pratica elimina tutte le configurazioni per cui almeno un errore in uno specifico input set relativo ha un valore maggiore di 0.95. Poiché infatti valori al di sopra di questa soglia possono essere molto erronei e fallaci negli approcci conseguenti di machine learning, viene data la possibilità di analizzare il dataset senza di essi.
  In questa seconda modalità l'algoritmo in realtà ha un comportamento molto simile a quello descritto precedentemente con la differenza che, per ogni campione nella lista delle configurazioni ordinate, ne viene scelta una casuale e se essa ha un errore non compatibile con quanto detto per almeno un input set, è scartata e un'altra configurazione randomica è scelta nello stesso campione. Se non ci sono configurazioni con errore sotto la suddetta soglia, viene riportato a runtime un errore, ma lo script continua a ricercare campioni indicativi per gli altri sottoinsiemi.

Le configurazioni risultanti sono poi utilizzate per dare in output due tipi di grafico:

1. a barre: sull'asse x vengono indicati gli input set e sull'asse y gli errori, descrivendo perciò l'errore in un input set specifico per una certa configurazione;
2. istogramma: descrive la distribuzione dell'errore nei dataset e quindi rappresenta gli errori sull'asse x e il numero degli input set nell'asse y. Questo grafico è quindi utile per considerare visivamente per ogni configurazione quanti input set hanno un errore in un range specifico (raggruppati in 10 contenitori a default).

Infine, sono calcolati anche dati statistici: per ogni configurazione infatti si riporta anche la media, la varianza e la deviazione standard. In più, è anche possibile visionare gli errori massimi e minimi.

- Verticalmente: questa rappresentazione considera il dataset in maniera verticale utilizzando un algoritmo di estrazione non molto diverso da quello descritto nell'approccio orizzontale.
  La differenza sostanziale è che qui ci si concentra però sugli input set: per ciascuno di essi vengono rappresentati gli errori relativi a n configurazioni.
  Questo script poi prende come argomenti il nome del dataset, il solito valore per indicare se filtrare o meno errori molto grandi e anche una variabile che indica i contenitori da utilizzare.
  I grafici risultanti sono perciò due:

  1. a barre: descrive la distribuzione degli errori considerando input set specifici. Più in dettaglio, i valori sull'asse x rappresentano tutte le n configurazioni, mentre quelli sull'asse y indicano gli errori. In questo modo quindi, è disponibile una utile visualizzazione di tutti gli errori in tutte le configurazioni per uno specifico input set;
  2. istogramma: descrive il numero di configurazioni in uno specifico range di errore (a default i contenitori sono 50, valore comunque modificabile durante il lancio dello script).
     L'asse x perciò rappresenta gli errori, mentre l'asse y indica il numero delle configurazioni.

Infine, anche qui vengono indicate informazioni statistiche come media, varianza, deviazione standard e valori di minimo e massimo.

- Configurazioni vicine: rappresentazione utile per considerare come gli errori relativi ai vicini siano distribuiti nei vari input set.
  In particolare, lo script prende come argomenti il nome del dataset, il solito valore di filtraggio, il numero delle configurazioni per cui fare i grafici e una lista di input set da considerare nell'analisi.
  L'ultimo valore è inserito specificamente per limitare il numero totale dei grafici trovati dal programma: infatti per ogni configurazione, sono calcolati $n+1$ grafici dove n è il numero degli input set da considerare (quindi al massimo 31 grafici per ogni configurazione).
  L'algoritmo principale inizialmente estrae i dati in maniera del tutto simile a quanto spiegato per gli approcci precedenti: il dataset è infatti letto riga per riga e viene costruita passo a passo una lista contenente tutti gli errori relativi agli specifici input set.
  Questi errori sono poi raggruppati per vicini: viene trovata la configurazione principale e poi messa in relazione alle configurazioni vicine.
  Infine, in maniera simile all'approccio orizzontale, le configurazioni sono scelte intelligentemente e dipendentemente dall'opzione di filtraggio impostata.
  Anche qui vengono quindi riportati dei grafici:

  1. un istogramma: rappresenta la distribuzione dell'errore medio dei vicini in ognuno dei 30 input set. Per fare ciò, vengono indicati gli errori medi sull'asse y e gli input set sull'asse x;
  2. una serie di istogrammi: si tratta di n grafici che considerano n differenti input set. Per ognuno di essi viene restituita una rappresentazione visiva della distribuzione dell'errore tra i diversi vicini. L'asse x perciò rappresenta le configurazioni vicine, mentre l'asse y fa riferimento agli errori.

  Similmente agli altri due approcci, anche qua si riportano dati statistici come media, varianza e deviazione standard, calcolati per ogni grafico.

Di seguito si riportano alcuni grafici più significativi presi da un sottoinsieme di benchmark tra quelli considerati e nello specifico quelli relativi a Convolution, Correlation, DWT e BlackScholes.
É importante notare che tutti i grafici utilizzano la scala logaritmica in

maniera da visualizzare al meglio gli errori che normalmente sono molto diversi tra di loro e composti da molte cifre decimali.

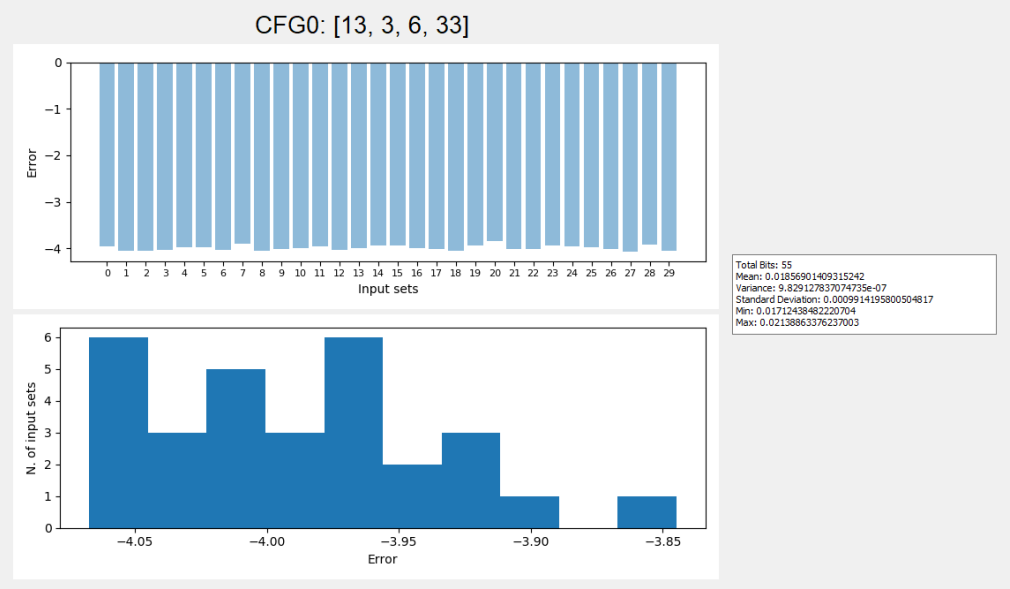


Figura 4.3: Configurazione a basso numero di bit totali (vista orizzontale del dataset Convolution in versione non filtrata)

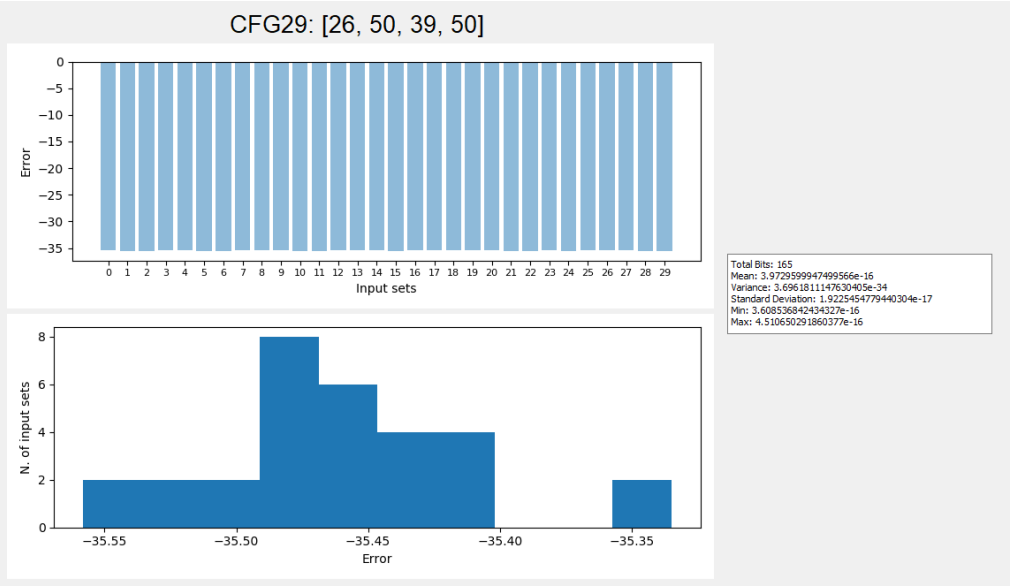


Figura 4.4: Configurazione ad alto numero di bit totali (vista orizzontale del dataset Convolution in versione filtrata)

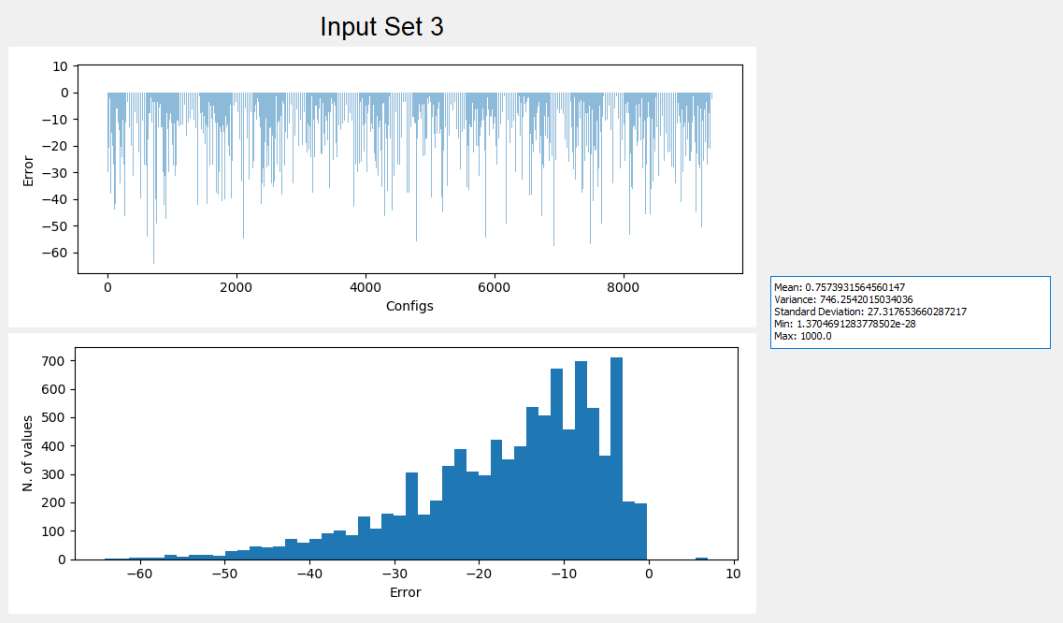


Figura 4.5: Analisi dell'input set 3 (vista verticale del dataset Convolution in versione non filtrata)

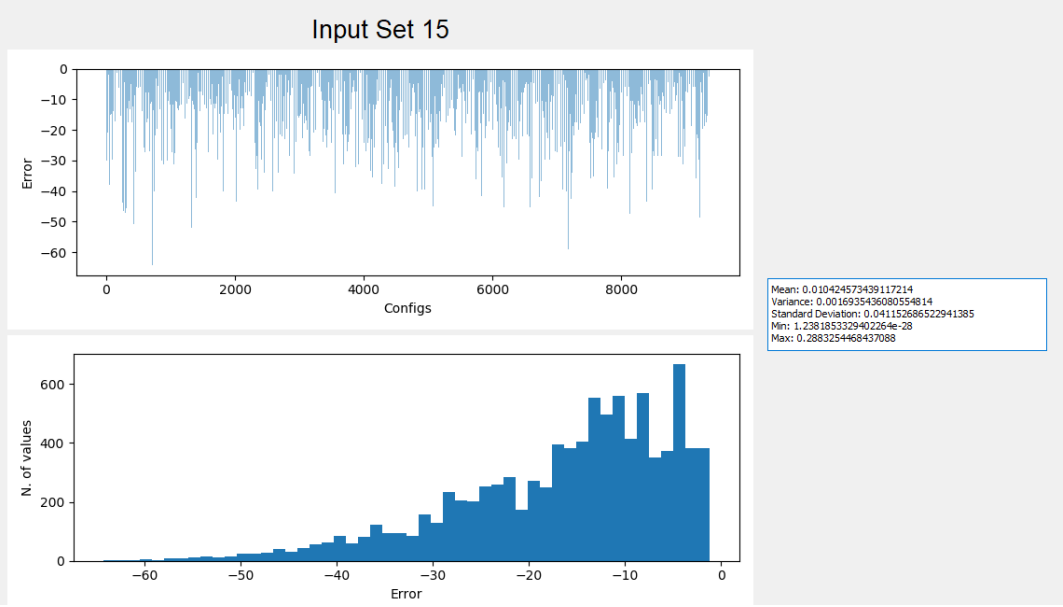


Figura 4.6: Analisi dell'input set 15 (vista verticale del dataset Convolution in versione filtrata)

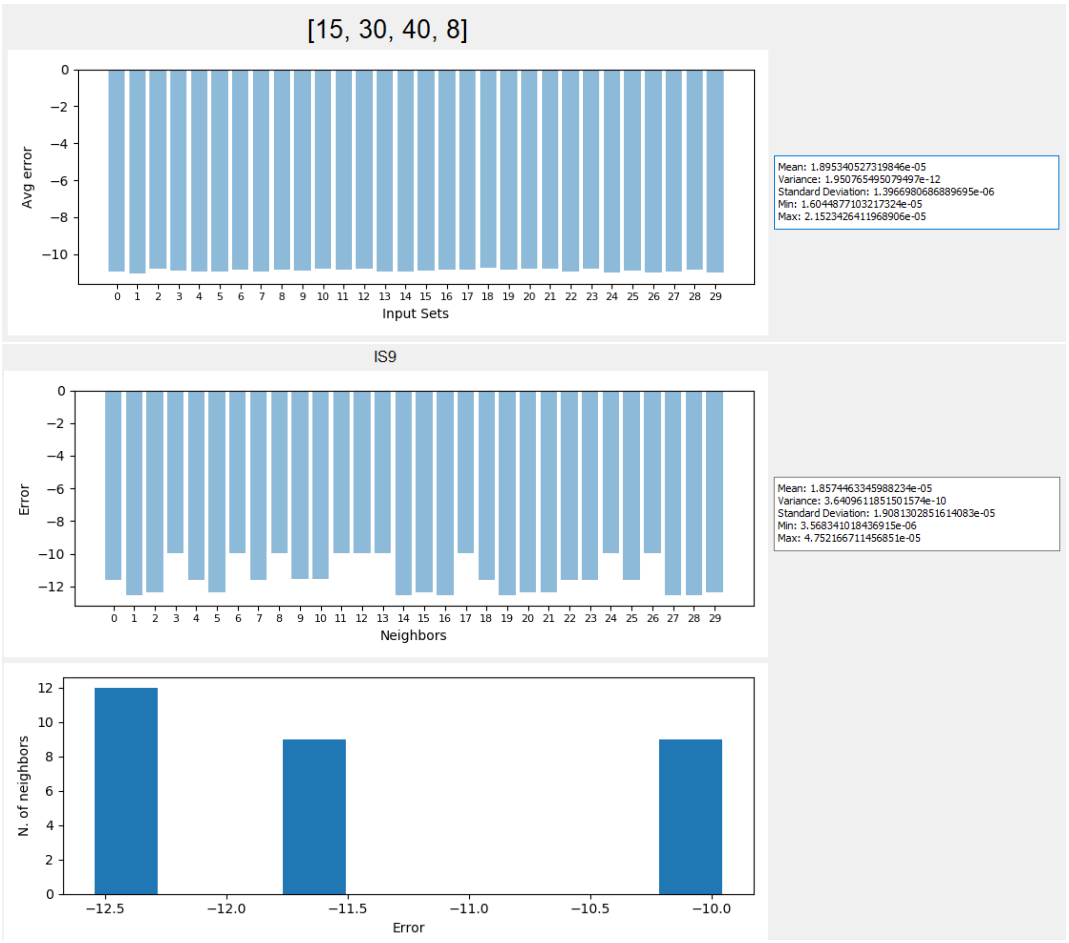
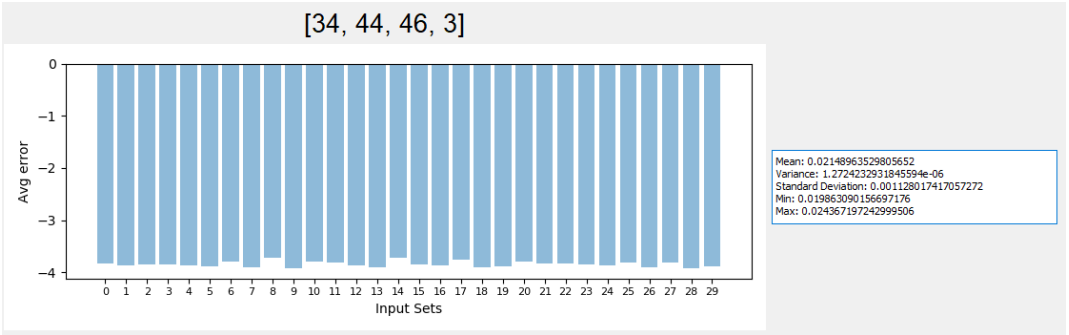


Figura 4.7: Grafici relativi ai vicini della configurazione indicata (dataset Convolution in versione non filtrata)



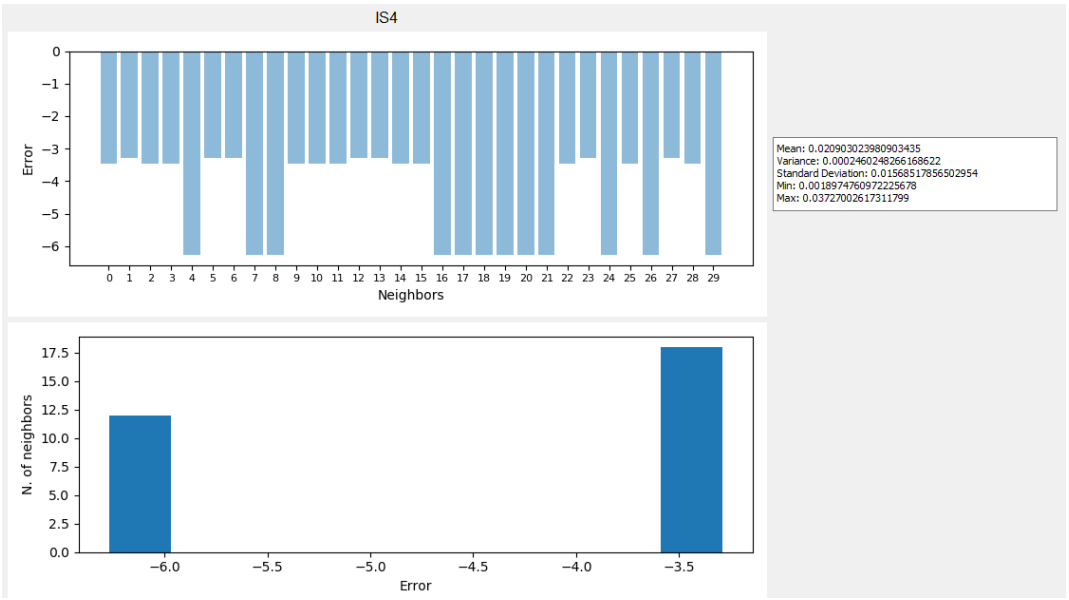


Figura 4.8: Grafici relativi ai vicini della configurazione indicata (dataset Convolution in versione filtrata)

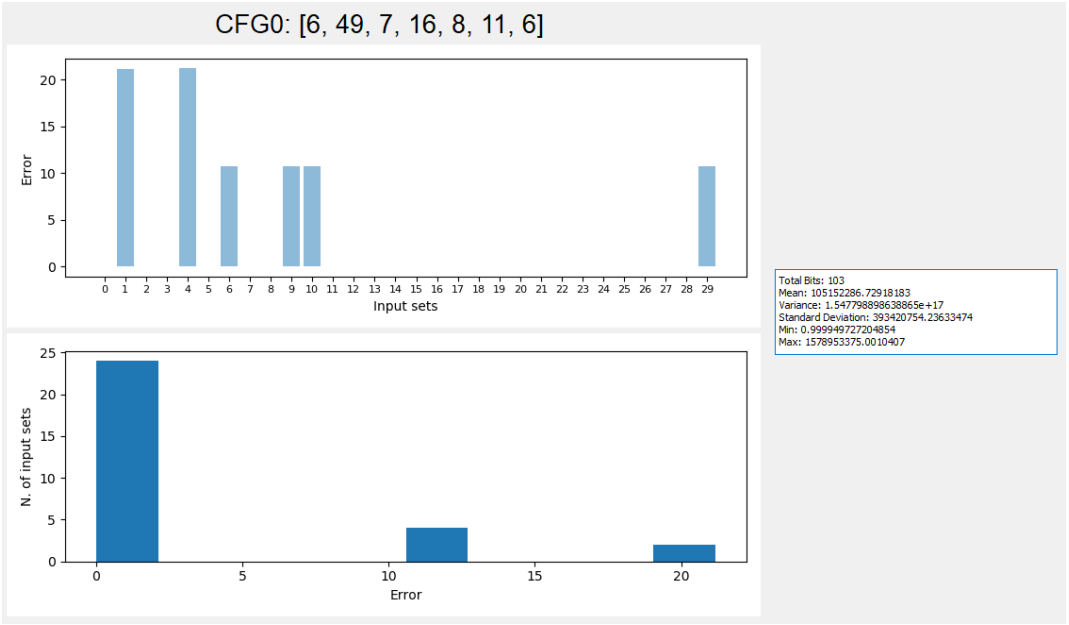


Figura 4.9: Configurazione a basso numero di bit totali (vista orizzontale del dataset Correlation in versione non filtrata)

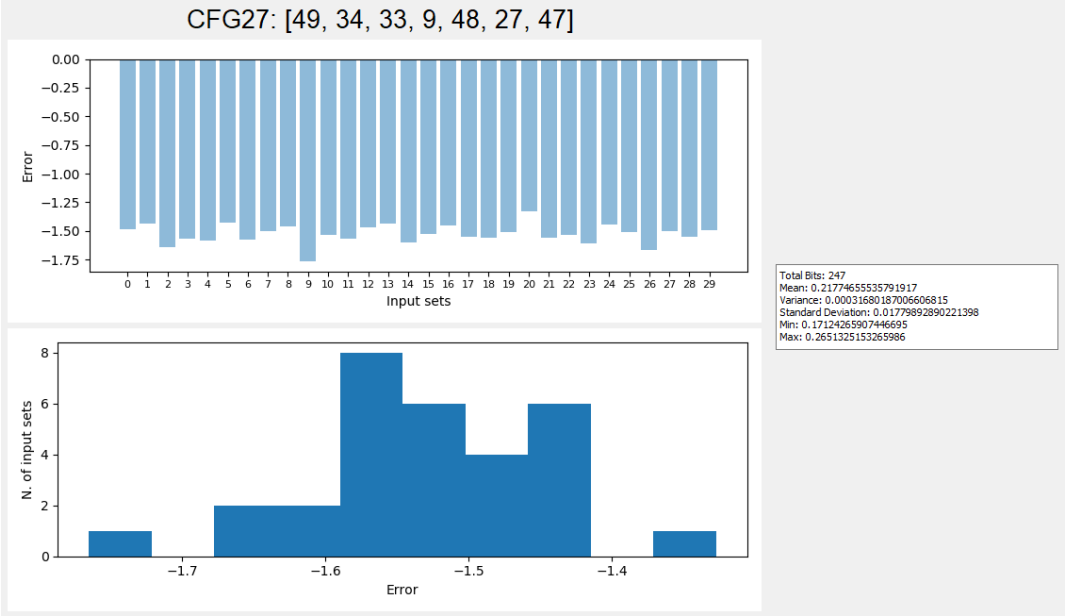


Figura 4.10: Configurazione ad alto numero di bit totali (vista orizzontale del dataset Correlation in versione filtrata)

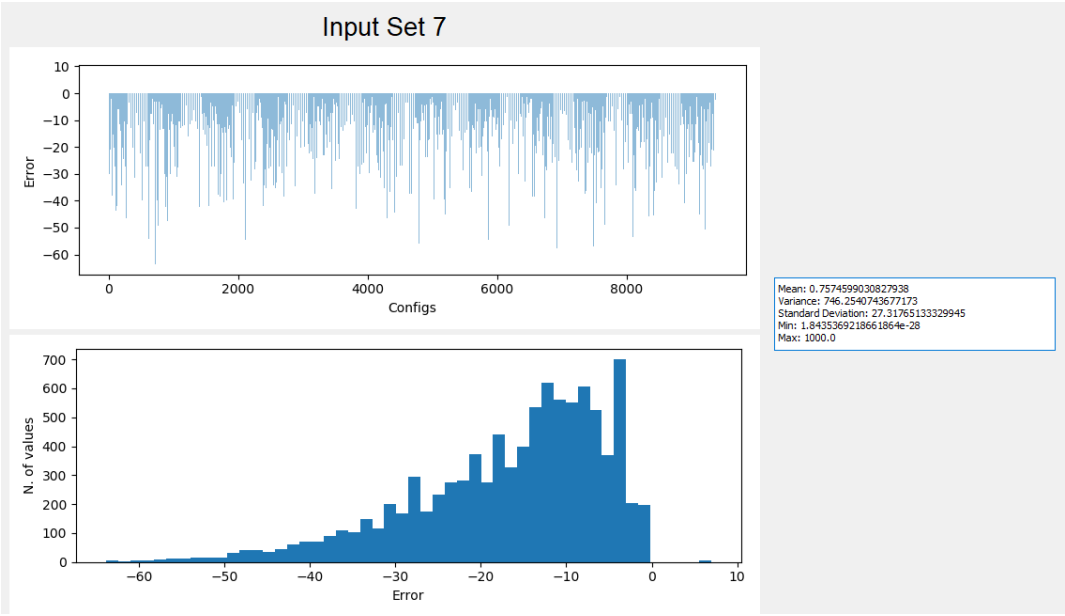


Figura 4.11: Analisi dell'input set 7 (vista verticale del dataset Correlation in versione non filtrata)

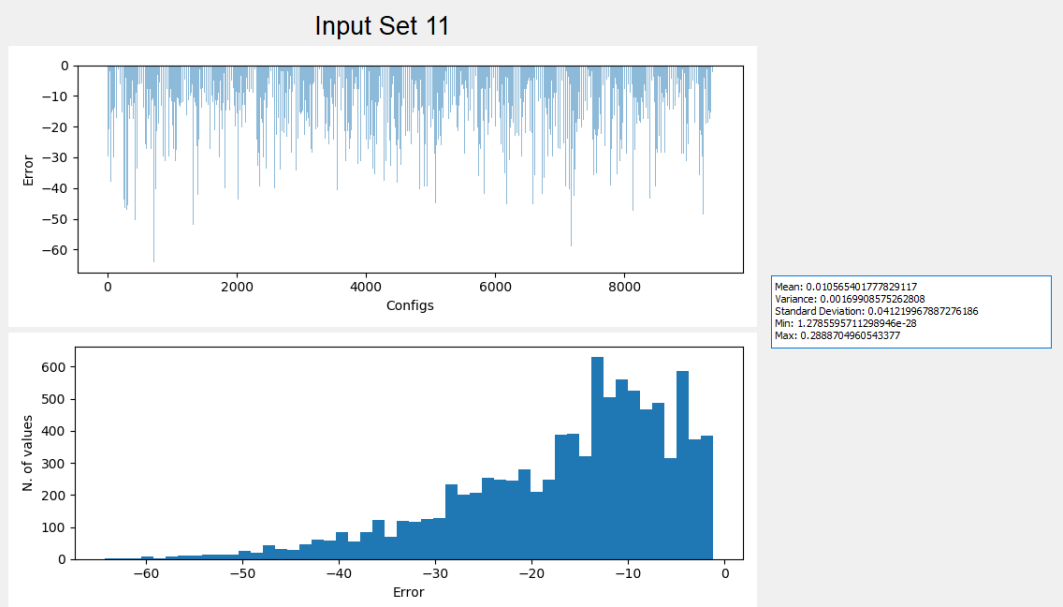


Figura 4.12: Analisi dell'input set 11 (vista verticale del dataset Correlation in versione filtrata)

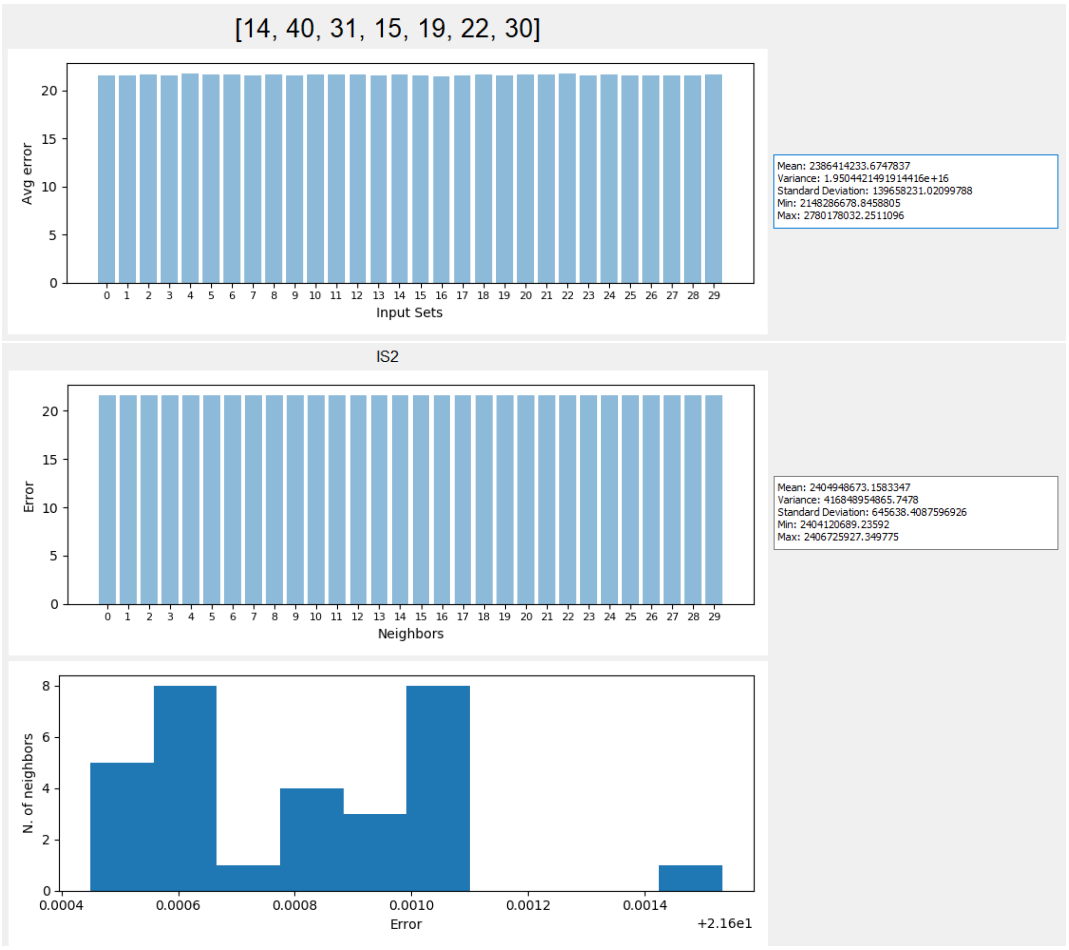
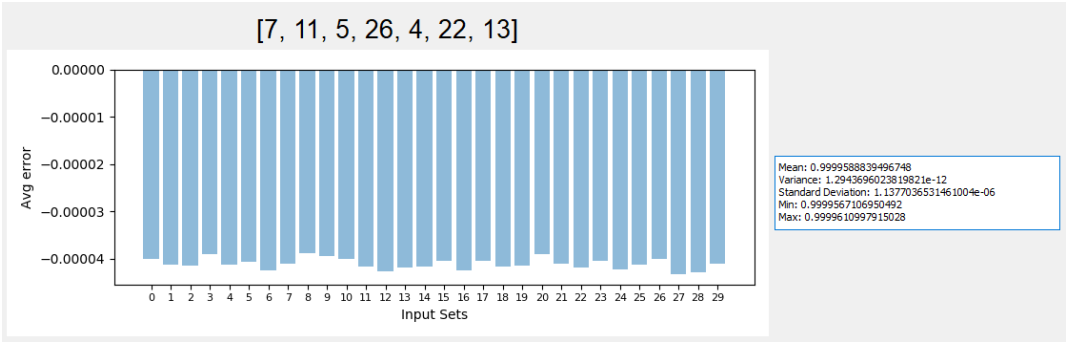


Figura 4.13: Grafici relativi ai vicini della configurazione indicata (dataset Correlation in versione non filtrata)



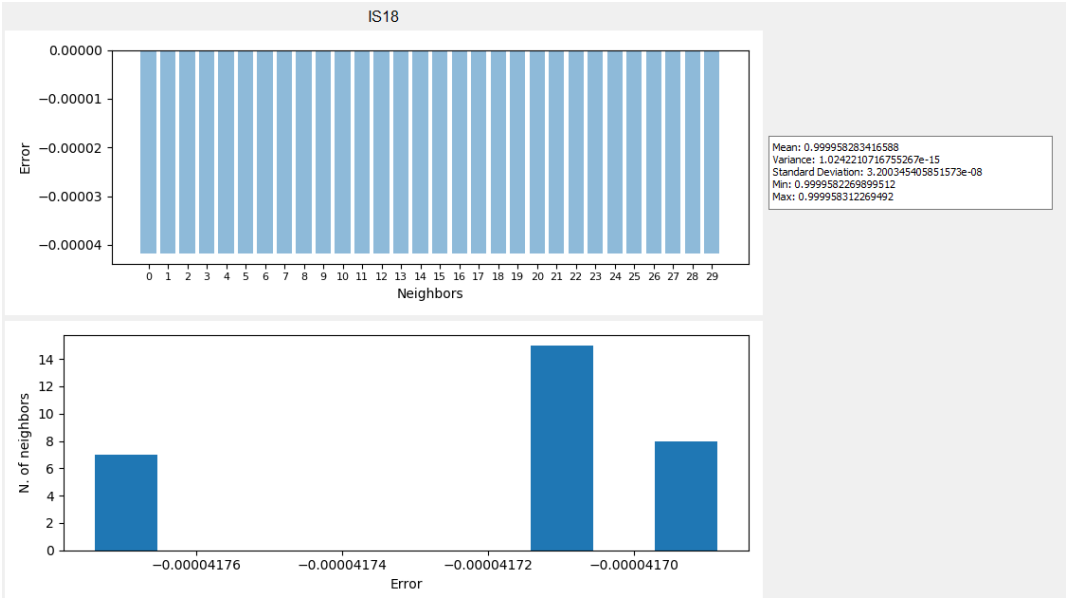


Figura 4.14: Grafici relativi ai vicini della configurazione indicata (dataset Correlation in versione filtrata)

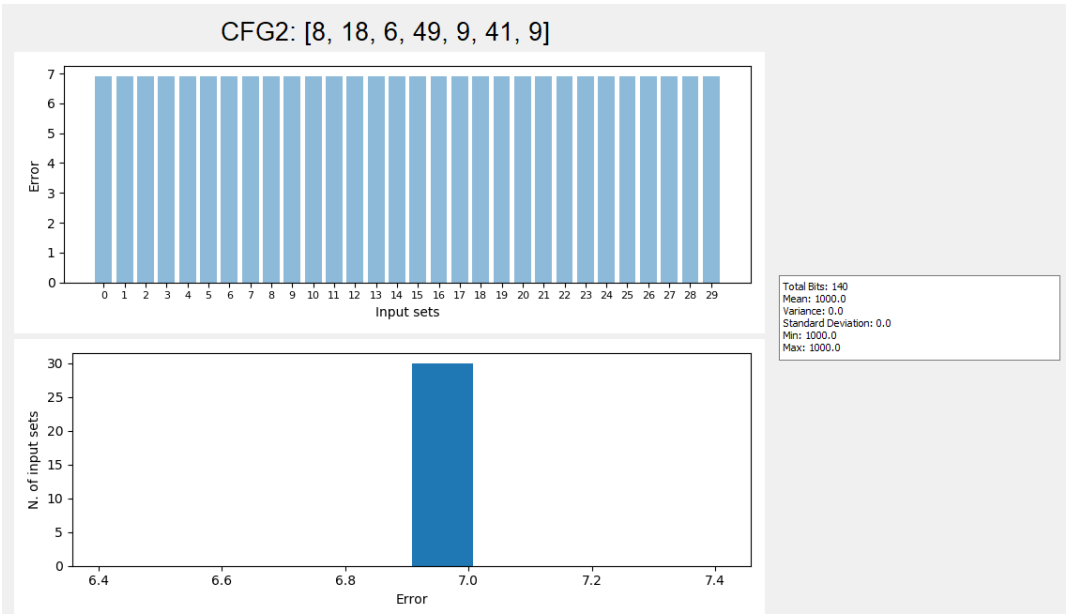


Figura 4.15: Configurazione a basso numero di bit totali (vista orizzontale del dataset DWT in versione non filtrata)

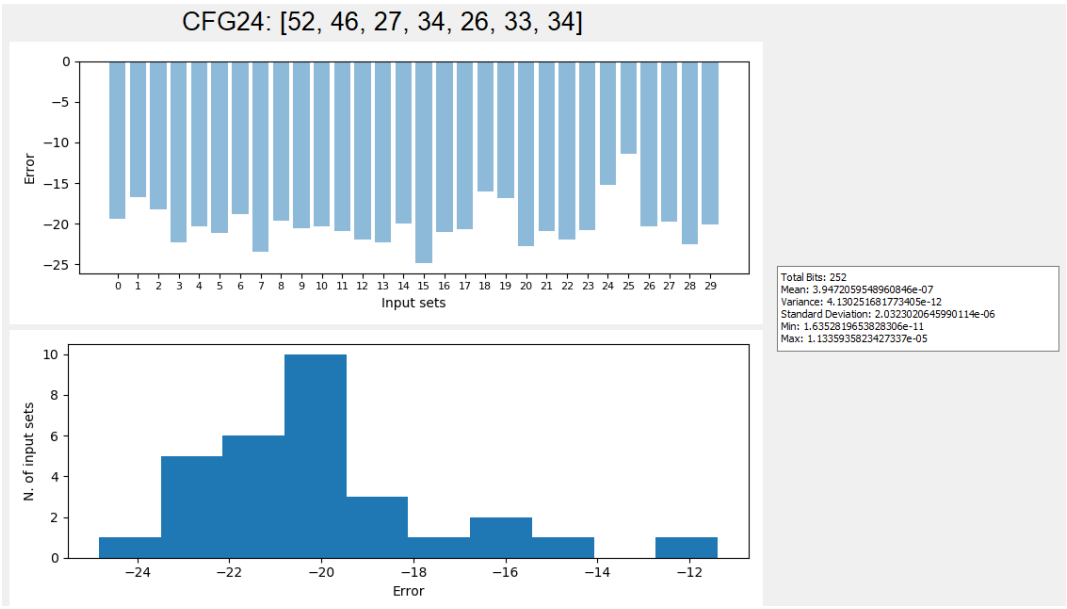


Figura 4.16: Configurazione ad alto numero di bit totali (vista orizzontale del dataset DWT in versione filtrata)

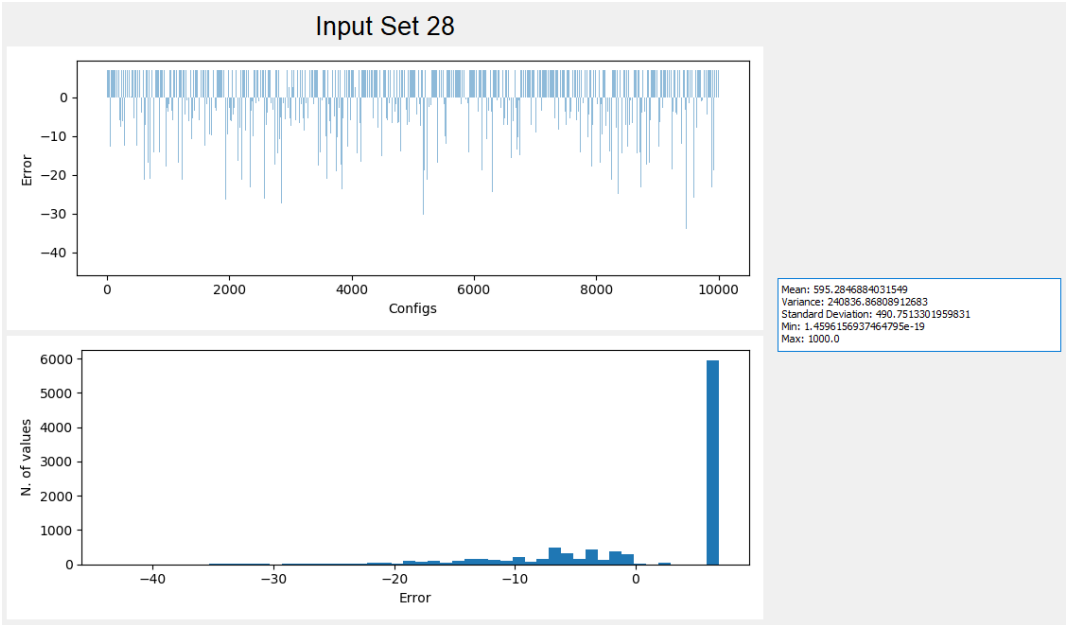


Figura 4.17: Analisi dell'input set 28 (vista verticale del dataset DWT in versione non filtrata)

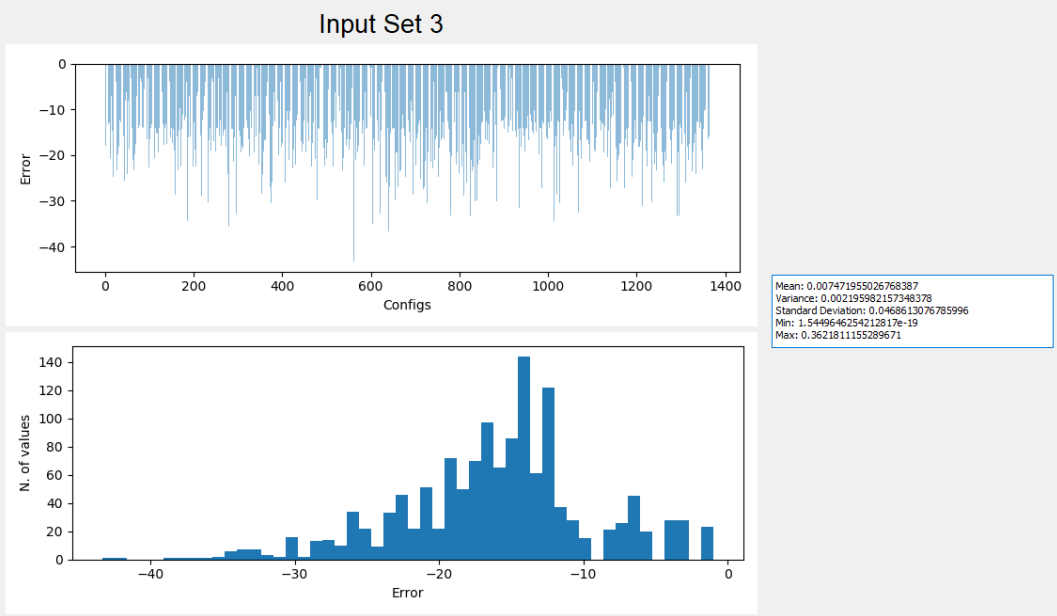


Figura 4.18: Analisi dell'input set 3 (vista verticale del dataset DWT in versione filtrata)

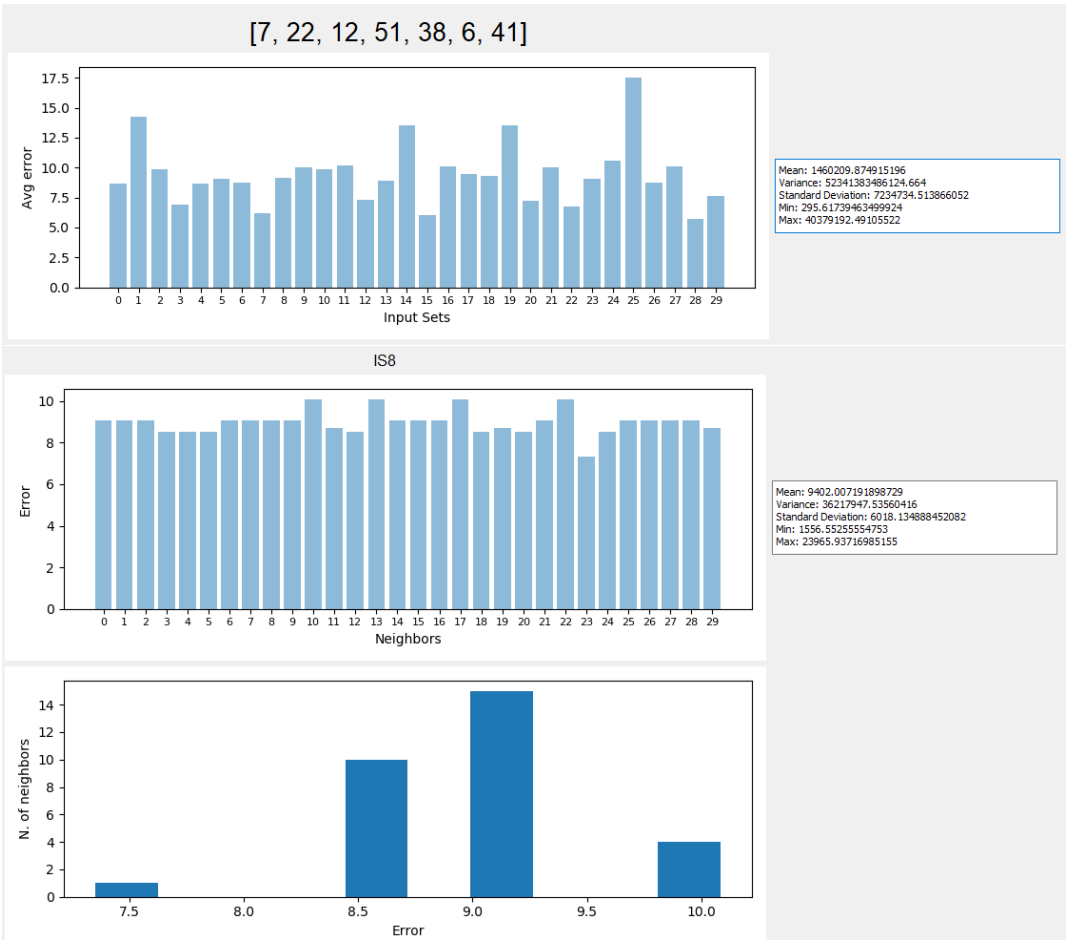
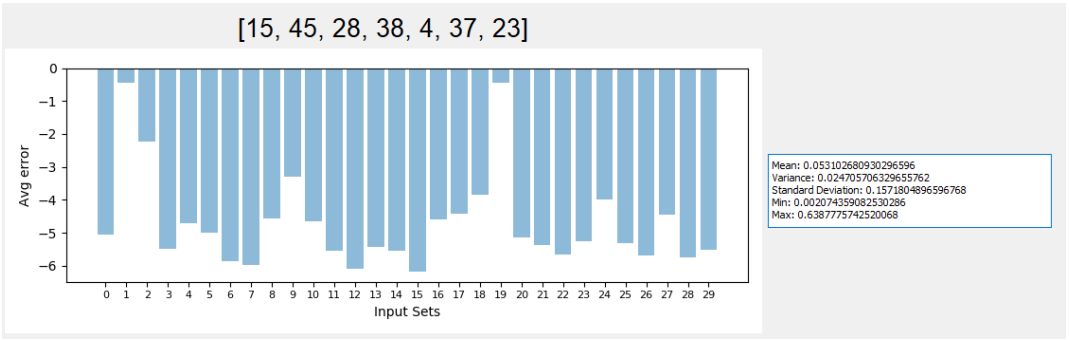


Figura 4.19: Grafici relativi ai vicini della configurazione indicata (dataset DWT in versione non filtrata)



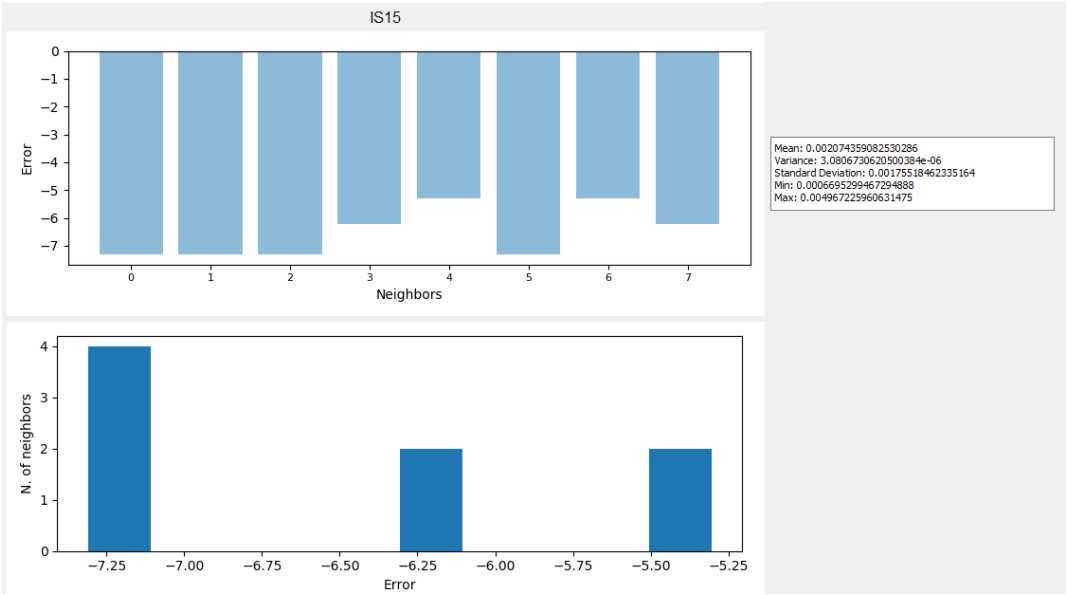


Figura 4.20: Grafici relativi ai vicini della configurazione indicata (dataset DWT in versione filtrata)

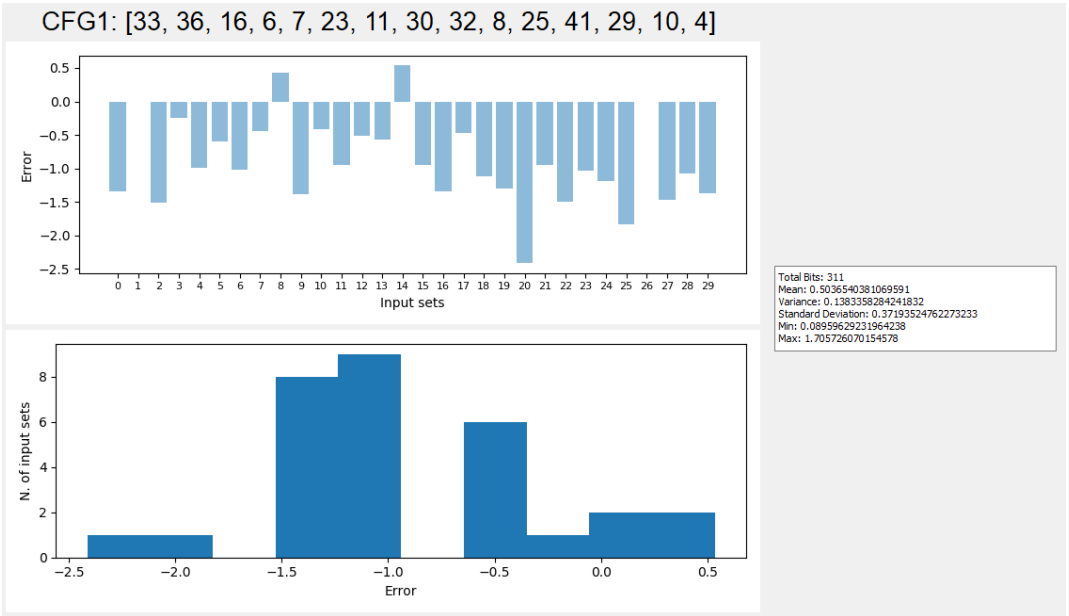


Figura 4.21: Configurazione a basso numero di bit totali (vista orizzontale del dataset BlackScholes in versione non filtrata)

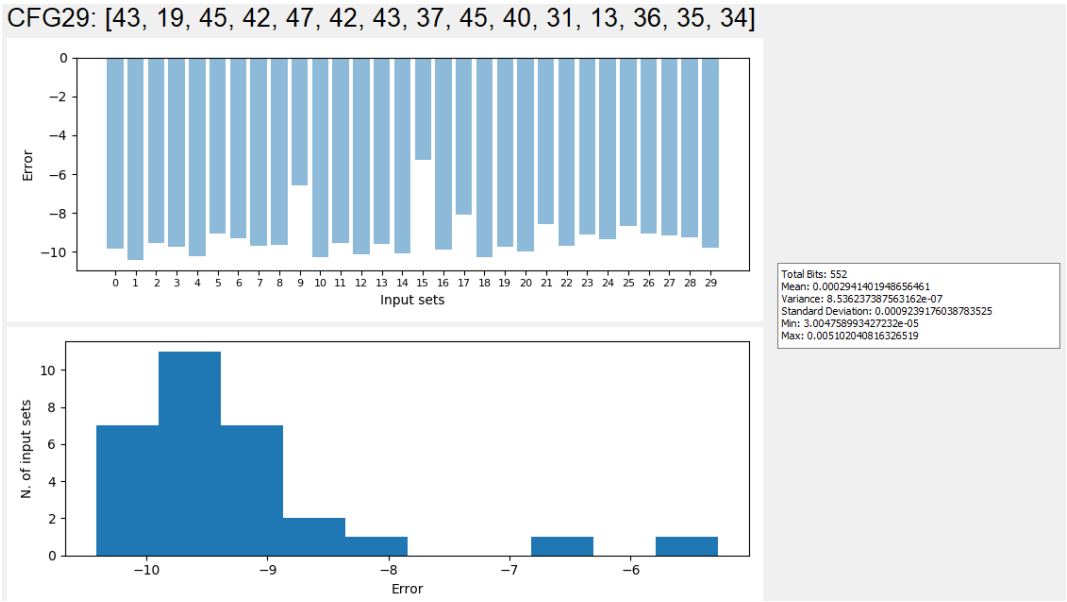


Figura 4.22: Configurazione ad alto numero di bit totali (vista orizzontale del dataset BlackScholes in versione filtrata)

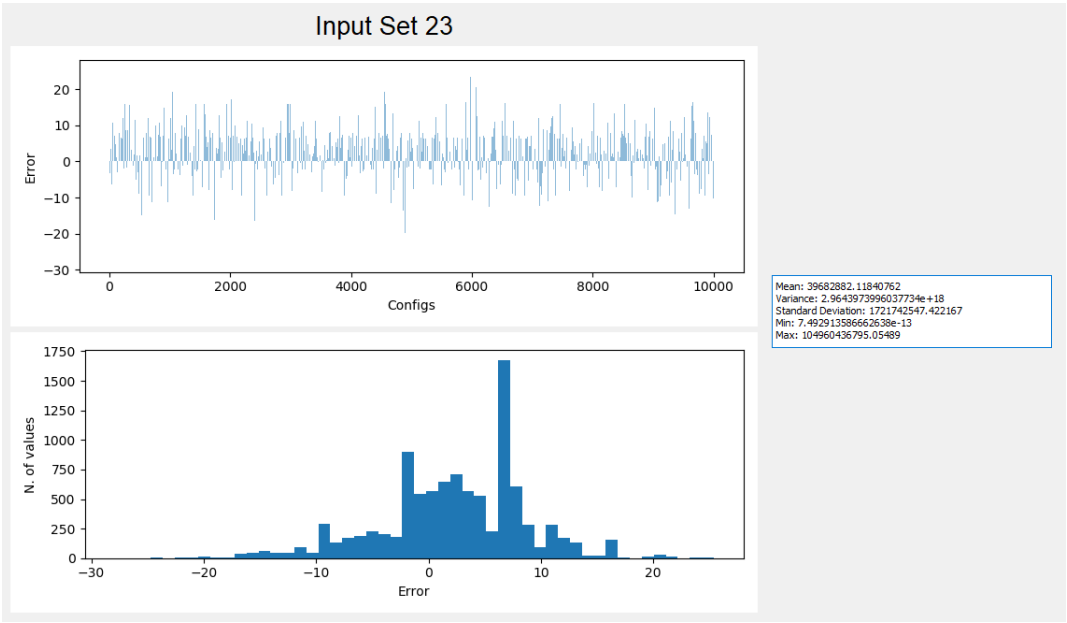


Figura 4.23: Analisi dell'input set 23 (vista verticale del dataset BlackScholes in versione non filtrata)

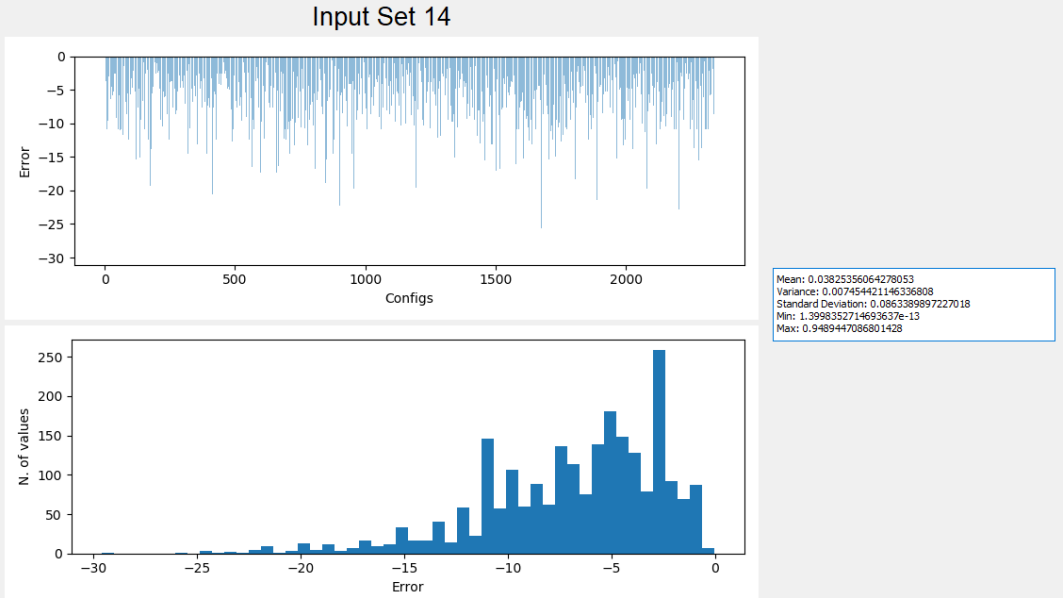


Figura 4.24: Analisi dell'input set 14 (vista verticale del dataset BlackScholes in versione filtrata)

In generale, considerando l'analisi orizzontale, emerge che l'errore di una configurazione varia molto a seconda dell'input set (di fatto non è per nulla uguale se l'input set è diverso). Inoltre poi, più i bit di precisione aumentano, più l'errore diminuisce.
Per quanto riguarda i grafici verticali, dagli istogrammi si nota invece che gli andamenti si mantengono tutti più o meno abbastanza simili.

4.2 Preprocessing dei dati

Prima di andare effettivamente ad implementare gli approcci di machine learning sui dataset generati per ogni benchmark, è necessario effettuare delle operazioni di raffinamento dei dati, per renderli più facilmente gestibili negli algoritmi implementati successivamente.
Sostanzialmente, come spiegato precedentemente, ogni dataset è una grande tabella costituita essenzialmente da due tipi di colonne, quelle relative alle variabili della $n-$esima configurazione e quelle inerenti invece agli errori calcolati sui diversi input set.
Il numero di colonne della prima tipologia è dipendente sempre dal particolare benchmark e quindi dalla cardinalità delle sue variabili, mentre la seconda tipologia presenta sempre 30 colonne, poiché tale è il numero degli input set considerati.

Dunque, è logico che operazioni di preprocessing dei dati debbano essere effettuate sia sulle variabili delle configurazioni, sia sugli errori.
In particolare, si è scelto di effettuare operazioni di normalizzazione sulla prima tipologia di dato e standardizzazione sull'altra, utilizzando le API di Scikit-learn.
La normalizzazione è un procedimento matematico utilizzato anche in statistica che permette di trasformare dei dati in un dataset con valori nell'intorno tra 0 e 1.
Più precisamente questa operazione è anche chiamata feature scaling ed è descritta come segue:

$$X_{new} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

La standardizzazione invece si occupa di trasformare i valori di un dataset in maniera che abbiano media pari a zero ($\mu = 0$) e deviazione standard pari ad uno ($\sigma = 1$):

$$X_{new} = \frac{X - \mu}{\sigma}$$

Generalmente la standardizzare viene effettuata se i dati che vengono trattati hanno naturalmente una disposizione gaussiana mentre la normalizzazione può essere usata anche senza assunzioni sulla distribuzione dei dati.
Tali operazioni sono state usate principalmente per avere dei miglioramenti sul training successivo dei modelli predittivi: effettuare standardizzazione e normalizzazione dei dati infatti porta ad ottenere una convergenza molto più rapida (ovvero modelli predittivi allenati molto più velocemente).

4.3 Regressione lineare

Il primo modello di machine learning introdotto come approccio alla risoluzione del problema è stato quello della regressione lineare ed in particolare, si è scelto di utilizzare il LinearRegressor messo a disposizione da Scikit-learn.
Essendo un approccio molto semplice e banale, se ne riporta il codice di seguito:

```
def linear_regressor(dataset, s_scaler, verbose=False):

  #Split the dataset into attributes (X) and labels (Y)
  var_columns = [col for col in dataset if col.startswith('var')]
  err_column = [col for col in dataset.columns if col not in var_columns]
  x = dataset[var_columns].values
  y = dataset[err_column].values
```

```
#80% of the dataset is our training set while 20% is our test set
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.2,
                                                    random_state=0)

#Train the model
regressor = LinearRegression()
regressor.fit(x_train, y_train)

#Predicted values
y_pred = regressor.predict(x_test)

return evaluate(y_test, y_pred, s_scaler, dataset[err_column])
```

Con successivi test si è scoperto che, valutando le potenzialità dei regressori su tutti i benchmark considerati, l'indice di $R2\_score$[1] è rimasto purtroppo costantemente su valori molto bassi.
Questo evidenzia il fatto che i dataset su cui lavorano i modelli di machine learning non sono lineari e anche che la funzione da imparare è troppo complessa per essere sintetizzata attraverso un regressore.
Di conseguenza, si è deciso di passare a modelli ML più complicati ma anche più precisi.

4.4 Deep learning

L'approccio di machine learning successivo è stato quello di creare dei modelli di rete neurale a diverse profondità e a diverso numero di nodi totali al fine di sperimentarne la precisione.
Utilizzando Keras come strumento principale e Tensorflow come backend, sono state esplorate diverse configurazioni.
In particolare, sono stati presi come modelli iniziali due reti neurali a cinque livelli, sperimentate con diverse configurazioni di nodi:

- $10x10x10x10x10$: 5 livelli, i primi 4 formati da $10 * n$ nodi (dove n è il numero delle features in input) e l'ultimo costituito da un solo nodo per effettuare la regressione;
- $100x100x100x100x100$: 5 livelli, i primi 4 formati da $100 * n$ nodi (dove n è il numero delle features in input) e l'ultimo costituito da un solo nodo per effettuare la regressione;

Con la stessa logica, si è passati a sperimentare modelli più complessi e con un tempo di training molto maggiore, prima a 10 livelli e poi a 20:

- $10x10x10x10x10x10x10x10x10x10$ (10 livelli);

[1] Si rimanda il lettore al capitolo 5 per eventuali approfondimenti su tale indicatore.

- $4x8x20x3x2x4x6x2x7$ (10 livelli);
- $2x2x2x2x2x2x2x2x2x2x2x2x2x2x2x2x2x2x2x2$ (20 livelli)
- $10x10x10x10x10x10x10x10x10x10x10x10x10x10x10x10x10x10x10x10$ (20 livelli)
- $5x9x15x4x7x6x3x8x85x47x32x4x9x21x2x14x36x5x7$ (20 livelli)

Inizialmente i modelli sono stati sperimentati utilizzando le risorse disponibili in locale effettuando una parallelizzazione in modo da dividere i task sui core del processore.
Tale approccio si è però rivelato totalmente inefficiente e molto lento data l'elevata capacità di calcolo richiesta per effettuare tutti i training dei modelli di deep learning. A pesare ulteriormente sulle risorse c'è stata poi la scelta di non prendere i risultati del primo modello allenato con la specifica configurazione, bensì quelli inerenti al miglior modello tra n modelli effettivamente allenati.
Questo perciò ha comportato il training di $7 * n$ modelli di deep learning (n è poi stato fissato a 3).
La conseguenza netta di tutto ciò è che non è stato possibile ottenere dei risultati sfruttando la macchina locale dati gli evidenti limiti computazionali, problema che di fatto ha reso necessario lo sviluppo di un altro script completamente distribuito da eseguire su D.A.V.I.D.E.
La scelta si è rivelata azzeccata, dato che le perfomance sono state nettamente migliori (grazie all'enorme potenza computazionale resa disponibile da D.A.V.I.D.E.) e i tempi di calcolo drasticamente diminuiti.
Più in dettaglio, è stato sviluppato uno script in grado di parallelizzare il training di diversi modelli su più GPU: ogni job eseguito su D.A.V.I.D.E. attraverso SLURM lavora infatti su un singolo input set di un singolo benchmark: questo comporta che, in linea teorica e avendo a disposizione n nodi su cui eseguire la computazione, potrebbero essere creati e allenati in parallelo n modelli di reti neurali.
I risultati di ognuno di essi poi verrebbe tranquillamente raccolto in un unico file di output.
Questo approccio vincente si è rivelato del tutto soddisfacente e ha permesso di fatto la distribuzione totale dello script e la parallelizzazione completa di diversi modelli di rete neurale, tutti allenati nello stesso momento (o quasi) ma su risorse diverse.
Di seguito, si riporta la creazione del modello generico di rete neurale:

```
#Define the model
pred_model = Sequential()
```

```
#Set the input layer
pred_model.add(Dense(int(n_attributes * layers_config.pop(0)),
                     activation='relu',
                     input_shape=input_shape))

#Set the hidden layers
for n_nodes in layers_config:
  pred_model.add(Dense(int(n_attributes * n_nodes), activation='relu'))

#Layer N is always one neuron with linear activation function
pred_model.add(Dense(1, activation='linear'))

#Set the optimizer
pred_model.compile(optimizer='adam', loss='mean_squared_error',
                   metrics=['mean_squared_error'])

#Set the callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10, min_delta=1e-5)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=5,
                              min_lr=1e-5, factor=0.2)

#Train the model!
history = pred_model.fit(x_train, y_train, epochs=EPOCHS,
                         shuffle=True, validation_split=0.1,
                         batch_size=BATCH_SIZE, verbose=1 if verbose else 0,
                         callbacks=[early_stopping, reduce_lr])

#Get the predictions
y_pred = pred_model.predict(x_test)
```

4.5 Introduzione dei vincoli dati dal grafo

Come si diceva nei primi capitoli di tale elaborato, al fine di introdurre maggiore precisione nei modelli predittivi, sarebbe opportuno considerare anche i vincoli dati dal grafo delle dipendenze tra le variabili, che di fatto esiste ed è diverso a seconda del benchmark utilizzato.
Di seguito vengono riportati a tal proposito due grafi indicativi, quello di Correlation, benchmark costituito da un numero limitato di variabili (7) e perciò anche da poche dipendenze totali e quello di BlackScholes, benchmark invece molto più complesso (15 variabili):

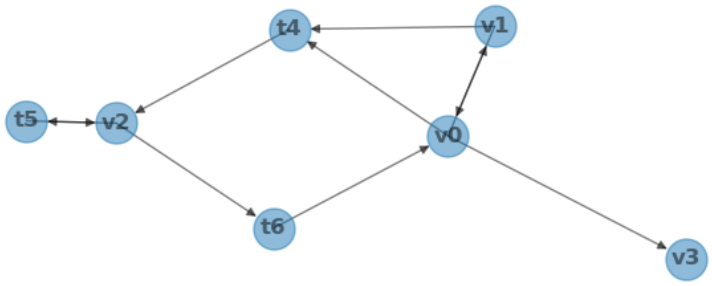


Figura 4.25: Grafo delle dipendenze relativo a Correlation

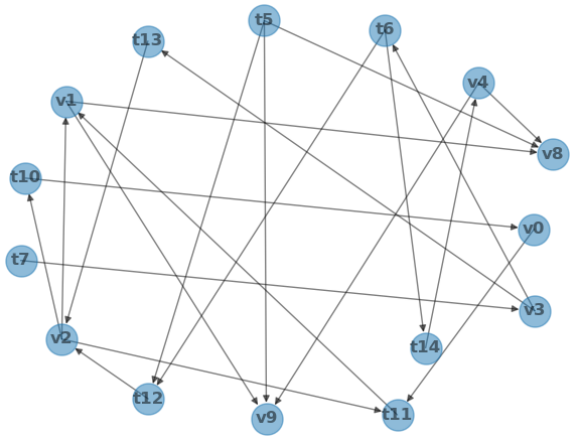


Figura 4.26: Grafo delle dipendenze relativo a BlackScholes

Di conseguenza, modelli con più legami tra le variabili saranno naturalmente più complessi e più facilmente inclini ad errori di predizione.
Per introdurre i vincoli dati dal grafo vengono proposte dunque due diverse modalità: inserire features aggiuntive nei dataset iniziali o creare reti ad-hoc pensate appositamente per lavorare con strutture a grafo, le GCN.

4.5.1 Features aggiuntive

Introdurre features aggiuntive nel training set significa essenzialmente considerare i vincoli nascosti nella struttura del grafo per arricchire la base di dati iniziale al fine di rendere le predizioni più accurate.
In particolare, lo script in Python appositamente creato per risalire alla struttura grafica delle dipendenze tra le variabili in un benchmark, viene ulteriormente considerato per estrarre tali vincoli in formato relazionale.
Si consideri ad esempio il grafo di Convolution qua riportato:

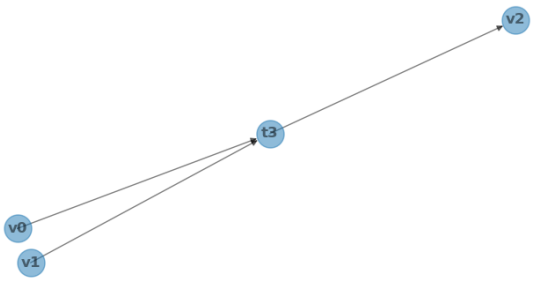


Figura 4.27: Grafo delle dipendenze relativo a Convolution

A seguire, si indicherà $t3$ come $v3$ ($t3$ è una variabile speciale usata da FlexFloat ma che coincide con $var3$ del dataset considerato).
Le informazioni estratte dal grafo sono sostanzialmente vincoli relazionali tra le features già esistenti: preso ad esempio l'arco da $v0$ in direzione di $v3$ ($v0 \rightarrow v3$), tale vincolo indica sostanzialmente che l'errore risultante del modello predittivo sarà più basso nel caso valesse in generale la relazione $v0 < v3$.
Questo si verifica poiché a runtime, FlexFloat andrà ad assegnare a $v0$ la precisione di $v3$ in uno dei passaggi di esecuzione del benchmark considerato. Di conseguenza, ovviamente il caso peggiore avverrà quando $v0 > v3$, poiché verrà effettuata un'operazione di casting che influirà negativamente sull'errore predetto.
In caso contrario, non ci saranno problemi poiché l'operazione di casting non avrà praticamente nessun effetto negativo dato che la precisione di $v0$ rimane comunque minore di quella di $v3$.
In particolare, questa relazione può essere rappresentata come una nuova feature che ne indica la differenza tra i due valori ($v3 - v0$): più tale valore sarà basso, più l'errore predetto sarà alto in quanto significherebbe di fatto adattare un numero ad alta precisione ad uno a bassa precisione.
Tornando all'esempio sopra menzionato, si consideri ad esempio il dataset (semplificato) iniziale:

	var0	var1	var2	var3	err
config0	27	45	35	40	9.42e-17
config1	42	23	4	10	0.1504
config2	7	52	45	4	0.0041

Dal grafo riportato in figura 4.27, vengono dedotte le seguenti relazioni:

- $v0 < v3$ da cui deriva la feature aggiuntiva $v3 - v0$;
- $v1 < v3$ da cui deriva la feature aggiuntiva $v3 - v1$;
- $v3 < v2$ da cui deriva la feature aggiuntiva $v2 - v3$.

Premesso che ai diversi vn del grafo corrispondono i diversi $varn$ nel dataset (cioè $v0 = var0$, $v1 = var2$ e $v3 = var3$), si otterrebbe così la nuova base di dati iniziale:

	var0	var1	var2	var3	var3-var0	var3-var1	var2-var3	err
config0	27	45	35	40	13	-5	-5	9.42e-17
config1	42	23	4	10	-32	-13	-6	0.1504
config2	7	52	45	4	-3	-48	41	0.0041

In particolare, in rosso vengono riportati gli attributi che contribuiscono ad un peggioramento dell'errore, mentre in verde sono evidenziati gli apporti positivi derivanti dal grafo.
Tramite questo approccio, vengono perciò creati dei dataset iniziali estesi attraverso le nuove features calcolate partendo dai grafi specifici dei benchmark considerati.
Sulla base di questo training set iniziale allargato, vengono infine allenati gli stessi modelli predittivi considerati nei sottocapitoli 4.3 e 4.4.
É infine opportuno notare la diversa complessità dei modelli: per ogni vincolo tra due variabili ne viene inserita una aggiuntiva al dataset iniziale e perciò più sono tali attributi, più vincoli devono essere rappresentati e più il modello diventa conseguentemente complesso.
A tal proposito, non ci sono molte differenze ad esempio nel training di una rete neurale relativa a Convolution con dataset arricchito rispetto alla sua versione standard (ci sono infatti solo 3 attributi in più da considerare), ma creare un modello di questo genere a partire da Black-Scholes (che si ricorda ha la cardinalità delle variabili pari a 15) porta sicuramente ad impatti computazionali molto più pesanti.

4.5.2 GCN

Andare ad introdurre dei vincoli inerenti al grafo utilizzando le graph convolutional networks significa di fatto utilizzare un modello di rete neurale del tutto nuova, non sperimentata nei passi precedenti del lavoro di tesi.
In particolare, utilizzando Spektral come framework, descritto esaustivamente nel sottocapitolo 3.4, si tratterà nella pratica di creare un modello compatibile con esso.
Più specificamente, è stato scelto di implementarne uno in modalità mista: questa decisione è stata presa poiché nel dominio considerato si ha sempre un grafo con topologia predefinita per ogni benchmark conside-

rato, ma anche attributi che cambiano.
Lo studio del dominio in questione ha perciò reso inadatta la modalità singola, dove si ha un singolo grafo con topologia ed attributi fissati ma anche quella a batch, dove esistono addirittura più grafi, ognuno con la propria topologia ma un insieme di differenti attributi.
Per creare il modello appena descritto, occorre quindi introdurre una matrice di adiacenza A e una lista di matrici degli attributi X.
Grazie a queste informazioni vengono creati perciò i cosiddetti graph convolutional layers, che effettuano le opportune trasformazioni tipiche delle GCN.
Infine, si costruisce il modello vero e proprio andando ad utilizzare un numero arbitrario di questi graph convolutional layers supportati anche da normalissimi layers Dense forniti da Keras.

Per rendere più chiaro il procedimento complessivo, si riporta di seguito un esempio basato sulla creazione di una GCN a partire da Convolution.
Considerando come grafo quello già presentato in figura 4.27 e come dataset di esempio quello riportato nel sottocapitolo precedente, si procede inizialmente a calcolare la matrice di adiacenza A e la lista di matrici degli attributi X.
Per quanto riguarda il primo passaggio, utilizzando le conoscenze teoriche discusse nel sottocapitolo 2.2.2, si otterrà:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

La lista delle matrici X invece si ottiene semplicemente andando a riordinare le features in ingresso alla rete.
Ad esempio, considerando che solamente le prime due configurazioni riportate nel dataset sopramenzionato faranno effettivamente parte del training set e non considerando le operazioni di standardizzazione e normalizzazione che comunque normalmente verranno eseguite, esse saranno rappresentate come:

$$X\_train = \begin{bmatrix} 27 & 45 & 35 & 40 \end{bmatrix}, \begin{bmatrix} 42 & 23 & 4 & 10 \end{bmatrix}$$

Effettuando un opportuno reshaping, il nuovo X_train compatibile con il modello GCN sarà perciò una lista di due matrici:

$$X\_train = \begin{bmatrix} 27 \\ 45 \\ 30 \\ 40 \end{bmatrix}, \begin{bmatrix} 42 \\ 23 \\ 4 \\ 10 \end{bmatrix}$$

In definitiva, a partire da queste conoscenze si costruisce perciò un nuovo modello di rete neurale che lavora anche con le informazioni estrapolate dal grafo, riassunte attraverso la matrice di adiacenza.
Andando poi a comparare i risultati derivanti dal training di differenti modelli di GCN a diversi layer e quindi a diversa complessità, si trova infine il modello di rete più adatto per il dominio considerato.

Si riportano di seguito i passaggi chiave delle API di Spektral e di Keras per realizzare una generica GCN:

```
#Split the dataset
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2, random_state=0)

#Features reshaping
X_train, X_test = X_train[..., None], X_test[..., None]

N = X_train.shape[-2]      # Number of nodes in the graphs
F = X_train.shape[-1]      # Node features dimensionality
n_out = y_train.shape[-1]  # Dimension of the target

#Extract the adjaceny matrix
adj = ed.getAdjacencyMatrix(benchmark_name)
A = sp.csr_matrix(adj).astype(np.float32)

#Apply filter
fltr = normalized_laplacian(A).astype(np.float32)

X_in = Input(shape=(N, F))
A_in = Input(tensor=sp_matrix_to_sp_tensor(fltr))

#Build the model
graph_conv = GraphConv(32, activation='elu', kernel_regularizer=l2(l2_reg),
                       use_bias=True)([X_in, A_in])
graph_conv = GraphConv(32, activation='elu', kernel_regularizer=l2(l2_reg),
                       use_bias=True)([graph_conv, A_in])
flatten = Flatten()(graph_conv)
fc = Dense(128, activation='relu')(flatten)
fc = Dense(32, activation='relu')(fc)
fc = Dense(8, activation='relu')(fc)
output = Dense(n_out, activation='linear')(fc)

model = Model(inputs=[X_in, A_in], outputs=output)
model.compile(optimizer='adam', loss='mean_squared_error',
              metrics=['mean_squared_error'])

#Set the callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=8, min_delta=1e-5)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', patience=10, min_lr=1e-5,
                              factor=0.2)

#Train the model
history = model.fit(X_train, y_train, batch_size=BATCH_SIZE, validation_split=0.1,
                    epochs=EPOCHS, verbose=1 if verbose else 0,
                    callbacks=[early_stopping, reduce_lr])

#Evaluate the model
y_pred = model.predict(X_test)
```

5 Risultati

In questo capitolo finale verranno raccolti sinteticamente i risultati dei vari modelli sperimentati nella fase progettuale dell'elaborato di tesi.
A tal propositivo, verranno perciò presentati degli indicatori di qualità per ogni approccio di machine learning adoperato e verranno eseguiti dei confronti per verificarne la soluzione migliore, i punti di forza e i punti carenti.
Inizialmente perciò si introducono brevemente i significati teorici di tali valori, utilizzati nelle comparazioni successive:

- RMSE (root mean square error): è una maniera standard di misurazione dell'errore di predizione in modelli di machine learning. Formalmente è definito come:

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

  dove $\hat{y}_1, \hat{y}_2, ..., \hat{y}_n$ sono i valori predetti, $y_1, y_2, ..., y_n$ sono i valori reali ed n è il numero delle osservazioni.
  Intuitivamente l'RMSE è un valore che misura la distanza tra il vettore dei valori predetti e quello dei valori reali ed occorre perciò che sia minimo per ottenere un modello preciso.
  Tutti i modelli predittivi descritti nella fase progettuale danno in output a tal proposito std_RMSE, il valore standardizzato di tale indice di accuratezza;

- MAE (mean absolute error): indica la discrepanza quadratica media fra i valori dei dati osservati ed i valori dei dati stimati.
  Formalmente è definito come:

  $$MAE = \frac{1}{n} \sum_{j=1}^{n} |\hat{y}_j - y_j|$$

dove $\hat{y}_1, \hat{y}_2, ..., \hat{y}_n$ sono i valori predetti, $y_1, y_2, ..., y_n$ sono i valori reali ed n è il numero delle osservazioni.
Trattandosi di una distanza, non esistono valori negativi e la ricerca del modello perfetto conduce inevitabilmente a trovare il valore di MAE più basso possibile.
Il grande vantaggio di questa metrica è la sua facile interpretazione, nonché la sua resistenza agli outliers.
Tutti i modelli predittivi descritti nella fase progettuale danno in output a tal proposito std_MAE, il valore standardizzato di tale indice di accuratezza, abs_MAE il valore in scala normale e $norm\_MAE$, il valore normalizzato;

- R-Squared: rappresenta la proporzione tra la variabilità dei dati e la correttezza del modello statistico utilizzato.
  Intuitivamente, misura quanto sono vicini i valori reali a quelli predetti, adoperando il contributo dei residui, intesi come la differenza tra questi due valori.

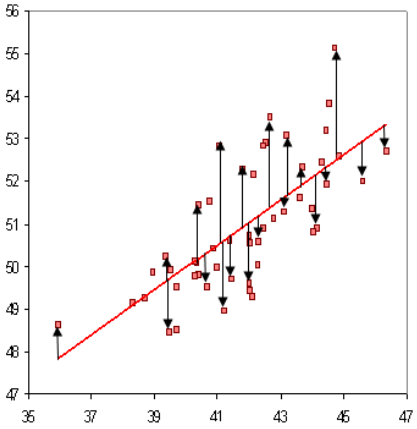


Figura 5.1: Rappresentazione grafica dei residui

  Tale valore è limitato superiormente ad 1, che rappresenta il miglior valore in assoluto e quindi la perfetta accuratezza del modello (i valori predetti sono identici a quelli reali).
  Tutti i modelli predittivi descritti nella fase progettuale danno in output a tal proposito $R2\_score$;

- errore medio: rappresenta il valor medio dell'errore ed è indicato con abs_mean;

- varianza: rappresenta la varianza dell'errore, indicata con abs_var.

5.1 Regressione lineare

Per quanto riguarda la regressione lineare, si presentano di seguito alcuni risultati relativi a diversi input set di Correlation e BlackScholes (non ci sono differenze molto evidenti tra gli input set dello stesso benchmark). I tempi di esecuzione dei vari modelli sono tutti molto bassi, in media sui 20-30 secondi.

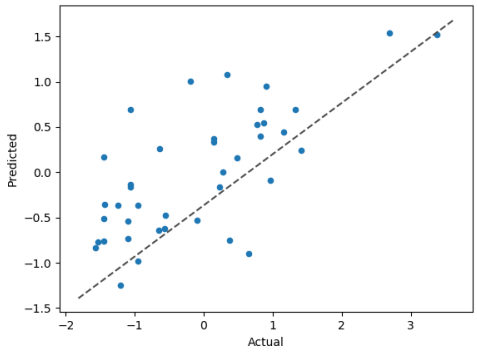


Figura 5.2: Scatter plot relativo a Correlation, dataset senza errori > 0.95 (input set 0)

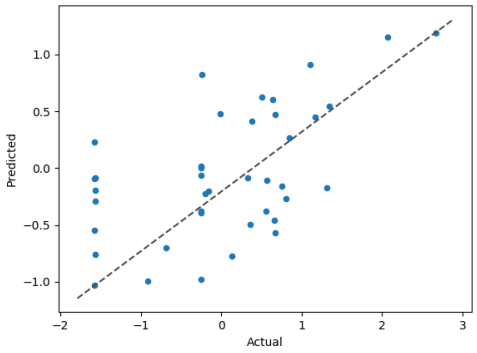


Figura 5.3: Scatter plot relativo a Correlation, dataset con tutti gli errori (input set 0)

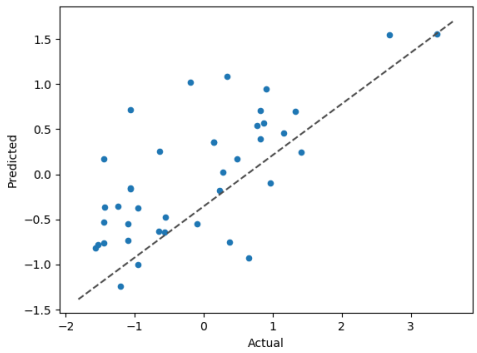


Figura 5.4: Scatter plot relativo a Correlation, dataset senza errori > 0.95 ed esteso per introdurre i vincoli del grafo (input set 0)

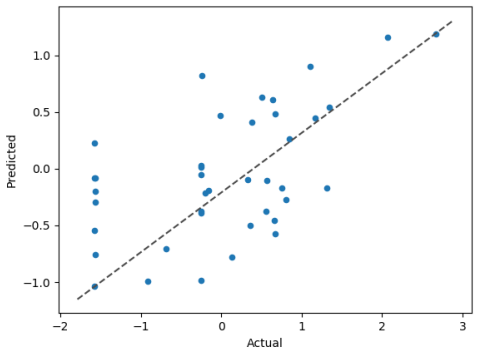


Figura 5.5: Scatter plot relativo a Correlation dataset con tutti gli errori ed esteso per introdurre i vincoli del grafo (input set 0)

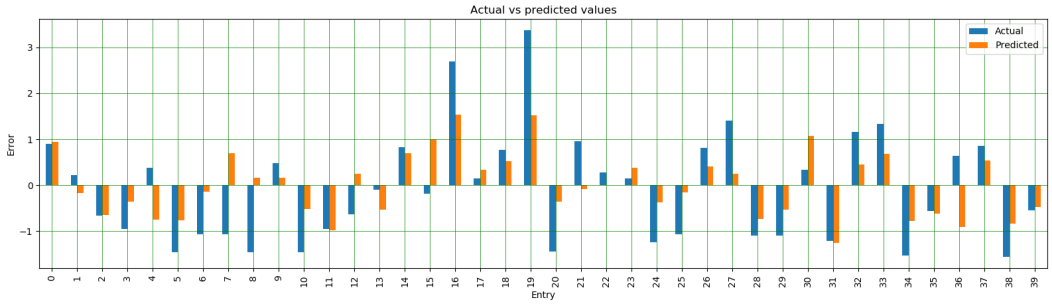


Figura 5.6: Confronto tra valori predetti e valori reali (Correlation, dataset senza errori > 0.95, input set 0)

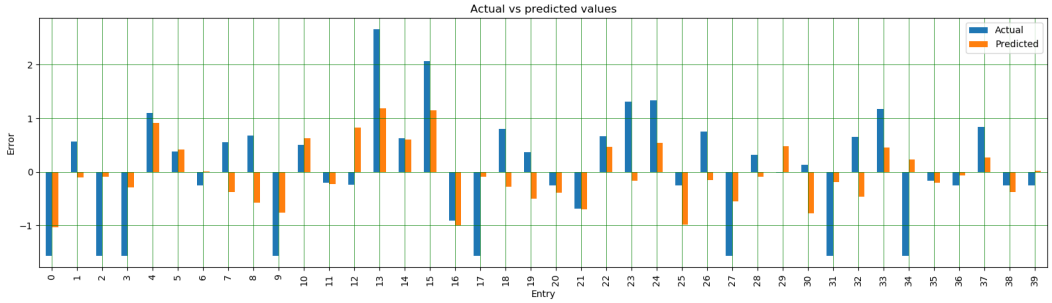


Figura 5.7: Confronto tra valori predetti e valori reali (Correlation, dataset con tutti gli errori, input set 0)

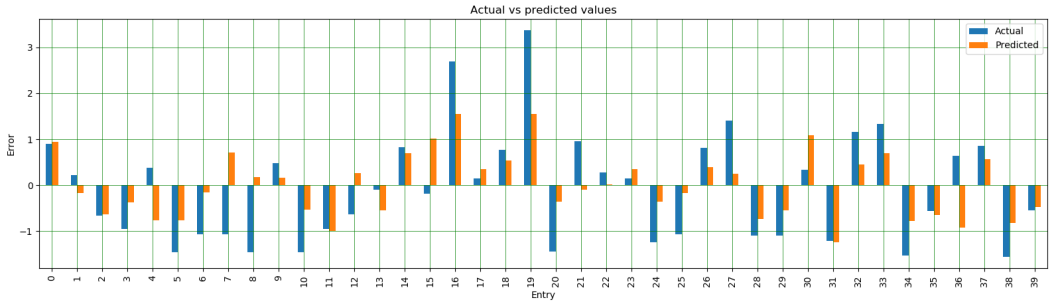


Figura 5.8: Confronto tra valori predetti e valori reali (Correlation, dataset senza errori > 0.95 ed esteso per introdurre i vincoli del grafo, input set 0)

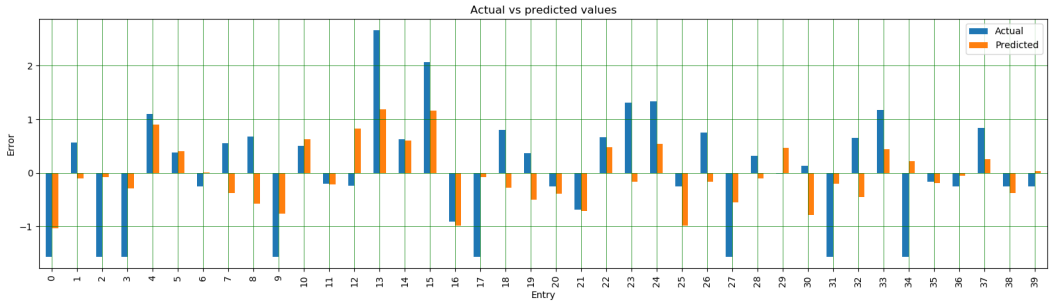


Figura 5.9: Confronto tra valori predetti e valori reali (Correlation, dataset con tutti gli errori ed esteso per introdurre i vincoli del grafo, input set 0)

Regression results

Benchmark: correlation

IS2		IS2 (Improved)		IS4		IS4 (Improved)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 6149 std_RMSE: 0.775 std_MAE: 0.628 abs_MAE: 5.81 norm_MAE: 0.388 abs_mean: 14.975 abs_var: 85.669 R2_score: 0.373	Total Entries: 9977 std_RMSE: 0.848 std_MAE: 0.699 abs_MAE: 11.306 norm_MAE: 2.46 abs_mean: 4.597 abs_var: 261.662 R2_score: 0.277	Total Entries: 6149 std_RMSE: 0.776 std_MAE: 0.629 abs_MAE: 5.821 norm_MAE: 0.389 abs_mean: 14.975 abs_var: 85.669 R2_score: 0.371	Total Entries: 9977 std_RMSE: 0.849 std_MAE: 0.699 abs_MAE: 11.313 norm_MAE: 2.461 abs_mean: 4.597 abs_var: 261.662 R2_score: 0.276	Total Entries: 5116 std_RMSE: 0.747 std_MAE: 0.611 abs_MAE: 5.875 norm_MAE: 0.367 abs_mean: 16.018 abs_var: 92.545 R2_score: 0.425	Total Entries: 9977 std_RMSE: 0.843 std_MAE: 0.711 abs_MAE: 12.079 norm_MAE: 5.417 abs_mean: 2.23 abs_var: 288.253 R2_score: 0.293	Total Entries: 5116 std_RMSE: 0.747 std_MAE: 0.612 abs_MAE: 5.885 norm_MAE: 0.367 abs_mean: 16.018 abs_var: 92.545 R2_score: 0.424	Total Entries: 9977 std_RMSE: 0.843 std_MAE: 0.712 abs_MAE: 12.084 norm_MAE: 5.419 abs_mean: 2.230 abs_var: 288.253 R2_score: 0.292

IS7		IS7 (Improved)		IS11		IS11 (Improved)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 4539 std_RMSE: 0.741 std_MAE: 0.598 abs_MAE: 5.753 norm_MAE: 0.348 abs_mean: 16.54 abs_var: 92.667 R2_score: 0.438	Total Entries: 9977 std_RMSE: 0.849 std_MAE: 0.701 abs_MAE: 11.245 norm_MAE: 4.207 abs_mean: 2.673 abs_var: 257.317 R2_score: 0.276	Total Entries: 4539 std_RMSE: 0.741 std_MAE: 0.597 abs_MAE: 5.751 norm_MAE: 0.348 abs_mean: 16.540 abs_var: 92.667 R2_score: 0.439	Total Entries: 9977 std_RMSE: 0.850 std_MAE: 0.701 abs_MAE: 11.251 norm_MAE: 4.209 abs_mean: 2.673 abs_var: 257.317 R2_score: 0.276	Total Entries: 6678 std_RMSE: 0.782 std_MAE: 0.636 abs_MAE: 5.77 norm_MAE: 0.398 abs_mean: 14.505 abs_var: 82.397 R2_score: 0.37	Total Entries: 9977 std_RMSE: 0.855 std_MAE: 0.684 abs_MAE: 9.744 norm_MAE: 1.438 abs_mean: 6.779 abs_var: 203.229 R2_score: 0.283	Total Entries: 6678 std_RMSE: 0.783 std_MAE: 0.636 abs_MAE: 5.771 norm_MAE: 0.398 abs_mean: 14.505 abs_var: 82.397 R2_score: 0.369	Total Entries: 9977 std_RMSE: 0.855 std_MAE: 0.684 abs_MAE: 9.746 norm_MAE: 1.438 abs_mean: 6.779 abs_var: 203.229 R2_score: 0.283

IS16		IS16 (Improved)		IS18		IS18 (Improved)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 5800 std_RMSE: 0.784 std_MAE: 0.63 abs_MAE: 5.881 norm_MAE: 0.383 abs_mean: 15.369 abs_var: 87.022 R2_score: 0.413	Total Entries: 9977 std_RMSE: 0.845 std_MAE: 0.696 abs_MAE: 11.058 norm_MAE: 2.47 abs_mean: 4.477 abs_var: 252.227 R2_score: 0.287	Total Entries: 5800 std_RMSE: 0.785 std_MAE: 0.631 abs_MAE: 5.882 norm_MAE: 0.383 abs_mean: 15.369 abs_var: 87.022 R2_score: 0.412	Total Entries: 9977 std_RMSE: 0.846 std_MAE: 0.697 abs_MAE: 11.069 norm_MAE: 2.472 abs_mean: 4.477 abs_var: 252.227 R2_score: 0.286	Total Entries: 6988 std_RMSE: 0.782 std_MAE: 0.63 abs_MAE: 5.714 norm_MAE: 0.405 abs_mean: 14.101 abs_var: 82.281 R2_score: 0.365	Total Entries: 9977 std_RMSE: 0.858 std_MAE: 0.684 abs_MAE: 10.327 norm_MAE: 1.633 abs_mean: 6.322 abs_var: 228.202 R2_score: 0.27	Total Entries: 6988 std_RMSE: 0.781 std_MAE: 0.630 abs_MAE: 5.713 norm_MAE: 0.405 abs_mean: 14.101 abs_var: 82.281 R2_score: 0.365	Total Entries: 9977 std_RMSE: 0.859 std_MAE: 0.685 abs_MAE: 10.343 norm_MAE: 1.636 abs_mean: 6.322 abs_var: 228.202 R2_score: 0.269

IS23		IS23 (Improved)		IS29		IS29 (Improved)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 6172 std_RMSE: 0.757 std_MAE: 0.615 abs_MAE: 5.701 norm_MAE: 0.382 abs_mean: 14.937 abs_var: 86.028 R2_score: 0.415	Total Entries: 9977 std_RMSE: 0.853 std_MAE: 0.694 abs_MAE: 10.519 norm_MAE: 1.909 abs_mean: 5.51 abs_var: 229.765 R2_score: 0.286	Total Entries: 6172 std_RMSE: 0.757 std_MAE: 0.615 abs_MAE: 5.700 norm_MAE: 0.382 abs_mean: 14.937 abs_var: 86.028 R2_score: 0.414	Total Entries: 9977 std_RMSE: 0.854 std_MAE: 0.695 abs_MAE: 10.529 norm_MAE: 1.911 abs_mean: 5.510 abs_var: 229.765 R2_score: 0.284	Total Entries: 5694 std_RMSE: 0.774 std_MAE: 0.625 abs_MAE: 5.863 norm_MAE: 0.379 abs_mean: 15.466 abs_var: 88.042 R2_score: 0.408	Total Entries: 9977 std_RMSE: 0.844 std_MAE: 0.702 abs_MAE: 11.712 norm_MAE: 3.346 abs_mean: 3.5 abs_var: 278.007 R2_score: 0.285	Total Entries: 5694 std_RMSE: 0.775 std_MAE: 0.627 abs_MAE: 5.882 norm_MAE: 0.380 abs_mean: 15.466 abs_var: 88.042 R2_score: 0.406	Total Entries: 9977 std_RMSE: 0.845 std_MAE: 0.703 abs_MAE: 11.714 norm_MAE: 3.346 abs_mean: 3.500 abs_var: 278.007 R2_score: 0.285

Figura 5.10: Confronto tra gli indici di qualità nei vari input set

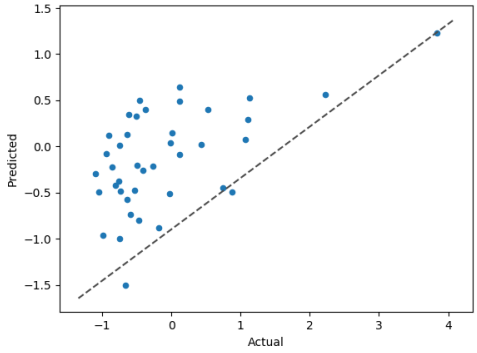


Figura 5.11: Scatter plot relativo a BlackScholes, dataset senza errori > 0.95 (input set 12)

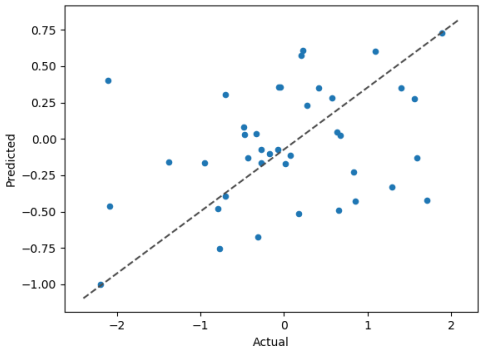


Figura 5.12: Scatter plot relativo a BlackScholes, dataset con tutti gli errori (input set 12)

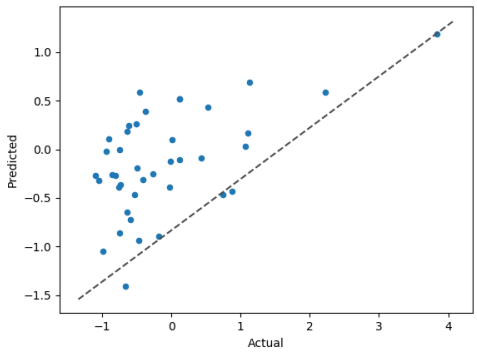


Figura 5.13: Scatter plot relativo a BlackScholes, dataset senza errori > 0.95 ed esteso per introdurre i vincoli del grafo (input set 12)

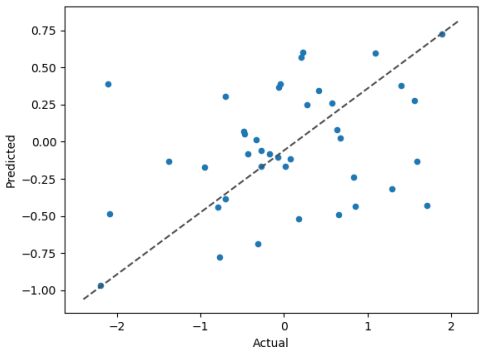


Figura 5.14: Scatter plot relativo a BlackScholes, dataset con tutti gli errori ed esteso per introdurre i vincoli del grafo (input set 12)

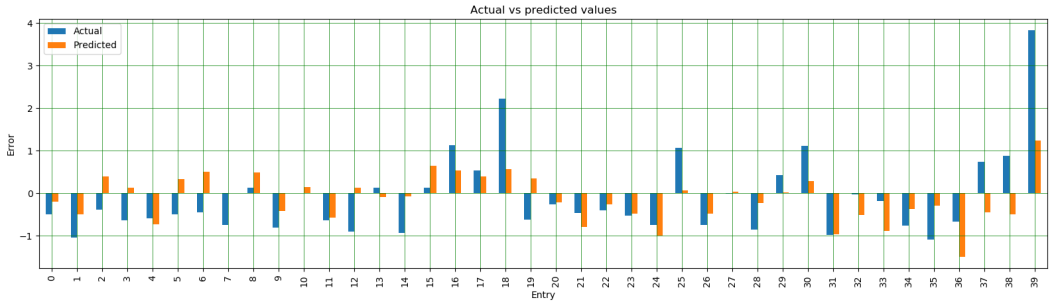


Figura 5.15: Confronto tra valori predetti e valori reali (BlackScholes, dataset senza errori > 0.95, input set 12)

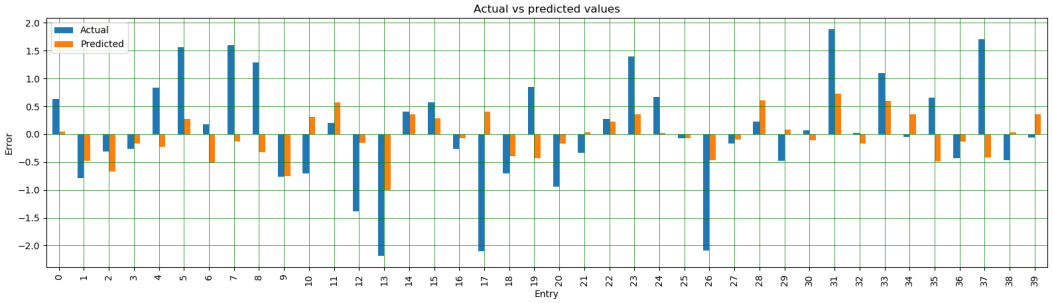


Figura 5.16: Confronto tra valori predetti e valori reali (BlackScholes, dataset con tutti gli errori, input set 12)

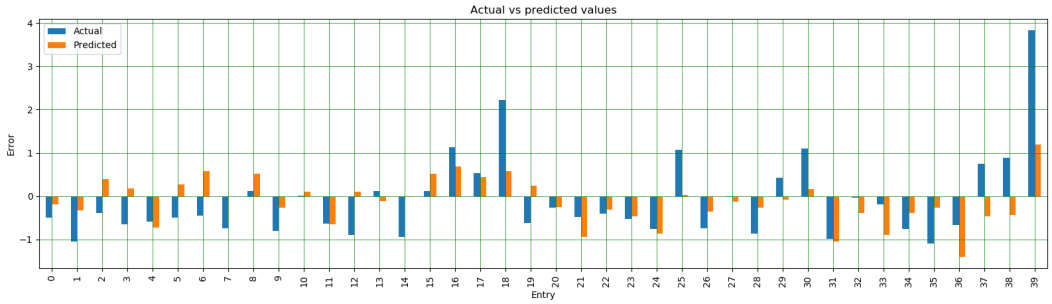


Figura 5.17: Confronto tra valori predetti e valori reali (BlackScholes, dataset senza errori > 0.95 ed esteso per introdurre i vincoli del grafo, input set 12)

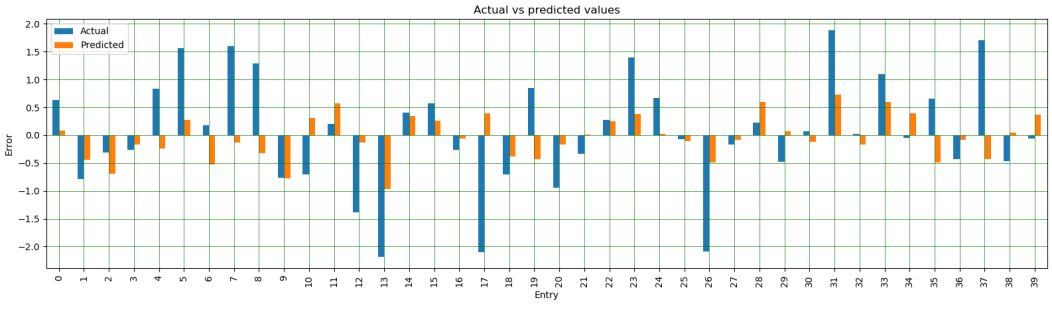


Figura 5.18: Confronto tra valori predetti e valori reali (BlackScholes, dataset con tutti gli errori ed esteso per introdurre i vincoli del grafo, input set 12)

Regression results

Benchmark: BlackScholes

IS1		IS1 (Improved)		IS3		IS3 (Improved)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 3119 std_RMSE: 0.888 std_MAE: 0.714 abs_MAE: 3.283 norm_MAE: 0.617 abs_mean: 5.317 abs_var: 21.119 R2_score: 0.135	Total Entries: 10000 std_RMSE: 0.892 std_MAE: 0.685 abs_MAE: 4.608 norm_MAE: -2.025 abs_mean: -2.275 abs_var: 45.271 R2_score: 0.195	Total Entries: 3119 std_RMSE: 0.889 std_MAE: 0.710 abs_MAE: 3.262 norm_MAE: 0.613 abs_mean: 5.317 abs_var: 21.119 R2_score: 0.134	Total Entries: 10000 std_RMSE: 0.891 std_MAE: 0.685 abs_MAE: 4.607 norm_MAE: -2.025 abs_mean: -2.275 abs_var: 45.271 R2_score: 0.197	Total Entries: 3132 std_RMSE: 0.877 std_MAE: 0.701 abs_MAE: 3.258 norm_MAE: 0.63 abs_mean: 5.176 abs_var: 21.634 R2_score: 0.201	Total Entries: 10000 std_RMSE: 0.879 std_MAE: 0.663 abs_MAE: 4.596 norm_MAE: -1.895 abs_mean: -2.426 abs_var: 48.039 R2_score: 0.195	Total Entries: 3132 std_RMSE: 0.881 std_MAE: 0.701 abs_MAE: 3.261 norm_MAE: 0.630 abs_mean: 5.176 abs_var: 21.634 R2_score: 0.194	Total Entries: 10000 std_RMSE: 0.879 std_MAE: 0.662 abs_MAE: 4.591 norm_MAE: -1.893 abs_mean: -2.426 abs_var: 48.039 R2_score: 0.196

IS7		IS7 (Improved)		IS13		IS13 (Improved)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 2857 std_RMSE: 0.923 std_MAE: 0.705 abs_MAE: 2.876 norm_MAE: 0.556 abs_mean: 5.172 abs_var: 16.655 R2_score: 0.153	Total Entries: 10000 std_RMSE: 0.884 std_MAE: 0.676 abs_MAE: 4.344 norm_MAE: -1.686 abs_mean: -2.576 abs_var: 41.317 R2_score: 0.194	Total Entries: 2857 std_RMSE: 0.929 std_MAE: 0.708 abs_MAE: 2.890 norm_MAE: 0.559 abs_mean: 5.172 abs_var: 16.655 R2_score: 0.142	Total Entries: 10000 std_RMSE: 0.883 std_MAE: 0.676 abs_MAE: 4.344 norm_MAE: -1.686 abs_mean: -2.576 abs_var: 41.317 R2_score: 0.194	Total Entries: 3344 std_RMSE: 0.882 std_MAE: 0.717 abs_MAE: 3.268 norm_MAE: 0.629 abs_mean: 5.192 abs_var: 20.752 R2_score: 0.172	Total Entries: 10000 std_RMSE: 0.883 std_MAE: 0.691 abs_MAE: 4.473 norm_MAE: -2.283 abs_mean: -1.96 abs_var: 41.865 R2_score: 0.206	Total Entries: 3344 std_RMSE: 0.886 std_MAE: 0.722 abs_MAE: 3.289 norm_MAE: 0.633 abs_mean: 5.192 abs_var: 20.752 R2_score: 0.166	Total Entries: 10000 std_RMSE: 0.882 std_MAE: 0.691 abs_MAE: 4.471 norm_MAE: -2.282 abs_mean: -1.960 abs_var: 41.865 R2_score: 0.208

IS21		IS21 (Improved)		IS25		IS25 (Improved)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 3331 std_RMSE: 0.92 std_MAE: 0.738 abs_MAE: 3.312 norm_MAE: 0.629 abs_mean: 5.265 abs_var: 20.145 R2_score: 0.161	Total Entries: 10000 std_RMSE: 0.881 std_MAE: 0.682 abs_MAE: 4.499 norm_MAE: -2.343 abs_mean: -1.92 abs_var: 43.464 R2_score: 0.207	Total Entries: 3331 std_RMSE: 0.921 std_MAE: 0.741 abs_MAE: 3.324 norm_MAE: 0.631 abs_mean: 5.265 abs_var: 20.145 R2_score: 0.160	Total Entries: 10000 std_RMSE: 0.881 std_MAE: 0.682 abs_MAE: 4.497 norm_MAE: -2.342 abs_mean: -1.920 abs_var: 43.464 R2_score: 0.208	Total Entries: 3331 std_RMSE: 0.874 std_MAE: 0.695 abs_MAE: 2.987 norm_MAE: 0.611 abs_mean: 4.888 abs_var: 18.476 R2_score: 0.171	Total Entries: 10000 std_RMSE: 0.894 std_MAE: 0.669 abs_MAE: 4.551 norm_MAE: -2.0 abs_mean: -2.276 abs_var: 46.236 R2_score: 0.186	Total Entries: 3331 std_RMSE: 0.878 std_MAE: 0.697 abs_MAE: 2.994 norm_MAE: 0.613 abs_mean: 4.888 abs_var: 18.476 R2_score: 0.163	Total Entries: 10000 std_RMSE: 0.894 std_MAE: 0.670 abs_MAE: 4.556 norm_MAE: -2.002 abs_mean: -2.276 abs_var: 46.236 R2_score: 0.187

IS26		IS26 (Improved)		IS28		IS28 (Improved)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 2818 std_RMSE: 0.916 std_MAE: 0.711 abs_MAE: 3.14 norm_MAE: 0.587 abs_mean: 5.347 abs_var: 19.516 R2_score: 0.195	Total Entries: 10000 std_RMSE: 0.883 std_MAE: 0.67 abs_MAE: 4.569 norm_MAE: -1.723 abs_mean: -2.652 abs_var: 46.463 R2_score: 0.196	Total Entries: 2818 std_RMSE: 0.923 std_MAE: 0.711 abs_MAE: 3.139 norm_MAE: 0.587 abs_mean: 5.347 abs_var: 19.516 R2_score: 0.182	Total Entries: 10000 std_RMSE: 0.882 std_MAE: 0.669 abs_MAE: 4.561 norm_MAE: -1.720 abs_mean: -2.652 abs_var: 46.463 R2_score: 0.198	Total Entries: 3408 std_RMSE: 0.898 std_MAE: 0.689 abs_MAE: 3.14 norm_MAE: 0.689 abs_mean: 4.56 abs_var: 20.765 R2_score: 0.143	Total Entries: 10000 std_RMSE: 0.889 std_MAE: 0.681 abs_MAE: 4.199 norm_MAE: -2.097 abs_mean: -2.002 abs_var: 37.972 R2_score: 0.207	Total Entries: 3408 std_RMSE: 0.898 std_MAE: 0.690 abs_MAE: 3.145 norm_MAE: 0.690 abs_mean: 4.560 abs_var: 20.765 R2_score: 0.142	Total Entries: 10000 std_RMSE: 0.889 std_MAE: 0.682 abs_MAE: 4.200 norm_MAE: -2.097 abs_mean: -2.002 abs_var: 37.972 R2_score: 0.209

Figura 5.19: Confronto tra gli indici di qualità nei vari input set

L'estensione del dataset per introdurre i vincoli dati dal grafo non porta purtroppo a nessun miglioramento particolare nel caso della regressione,

come si nota anche dai relativi scatter plots che sono praticamente identici nei casi di dataset esteso e dataset normale.
Inoltre, gli indici di qualità sono del tutto pessimi: questo deriva essenzialmente dalla non linearità dei dataset considerati e perciò dall'impossibilità di rappresentare correttamente la relazione cercata tra precisione dei bit e qualità del risultato tramite dei modelli di regressione.

5.2 Deep learning

In questa sezione vengono riportati alcuni risultati sperimentali per quanto riguarda l'approccio di deep learning ed in particolare vengono presi in considerazione quelli riguardanti i benchmark di Convolution e di Jacobi.
Vengono testati prima i modelli $10n * 10n * 10n * 10n * 2$, quindi più semplici, a 5 layer, per poi passare a quelli più complessi a 20 layer con la topologia $5n*9n*15n*4n*7n*6n*3n*8n*85n*47n*32n*4n*9n*21n*2n*14n*36n*5n*7n*2$ (si ricorda che n rappresenta il numero di variabili, 4 per Convolution e 25 per Jacobi).

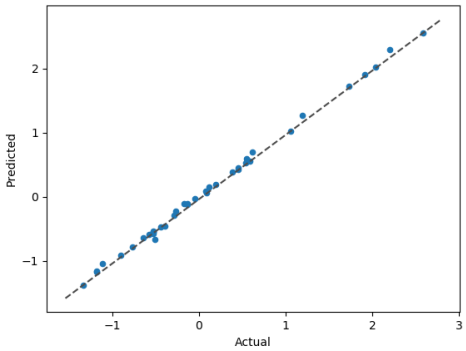


Figura 5.20: Scatter plot relativo a Convolution, dataset senza errori > 0.95 (input set 26), 5 layers

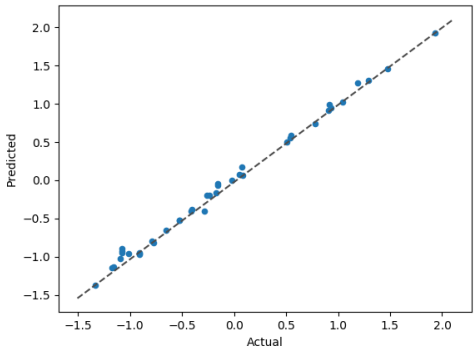


Figura 5.21: Scatter plot relativo a Convolution, dataset con tutti gli errori (input set 26), 5 layers

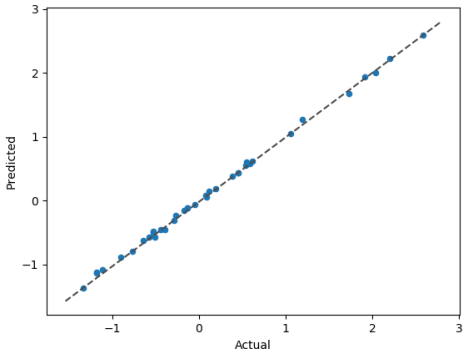


Figura 5.22: Scatter plot relativo a Convolution, dataset senza errori > 0.95 ed esteso per introdurre i vincoli del grafo (input set 26), 5 layers

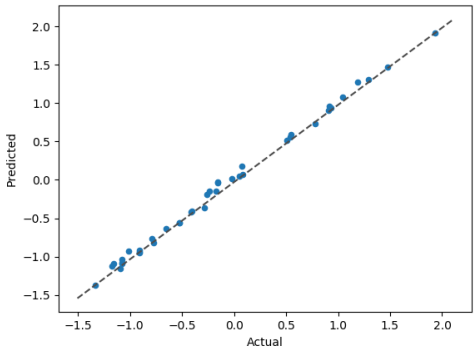


Figura 5.23: Scatter plot relativo a Convolution, dataset con tutti gli errori ed esteso per introdurre i vincoli del grafo (input set 26), 5 layers

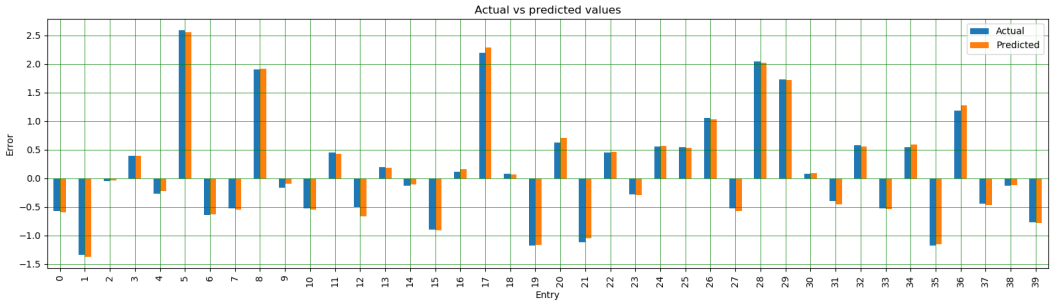


Figura 5.24: Confronto tra valori predetti e valori reali (Convolution, dataset senza errori > 0.95, input set 26, 5 layers)

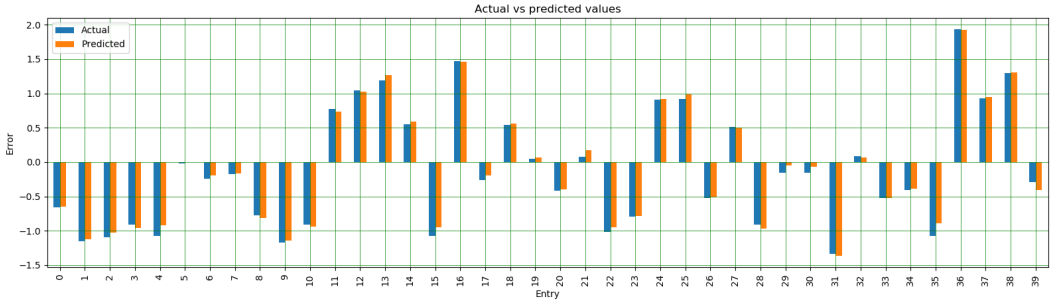


Figura 5.25: Confronto tra valori predetti e valori reali (Convolution, dataset con tutti gli errori, input set 26, 5 layers)

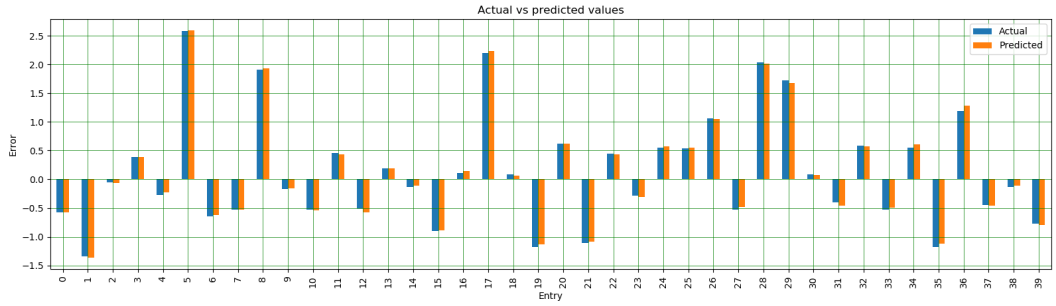


Figura 5.26: Confronto tra valori predetti e valori reali (Convolution, dataset senza errori > 0.95 ed esteso per introdurre i vincoli del grafo, input set 26, 5 layers)

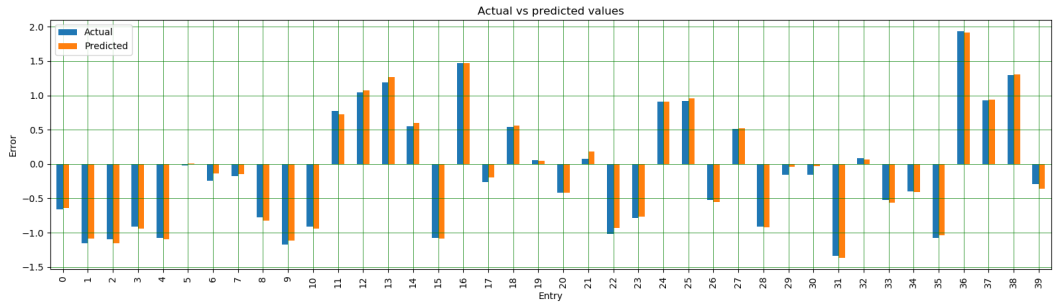


Figura 5.27: Confronto tra valori predetti e valori reali (Convolution, dataset con tutti gli errori ed esteso per introdurre i vincoli del grafo, input set 26, 5 layers)

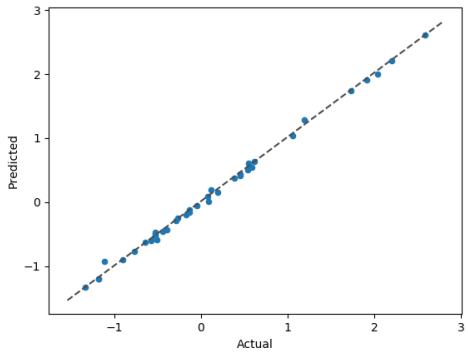


Figura 5.28: Scatter plot relativo a Convolution, dataset senza errori > 0.95 (input set 26), 20 layers

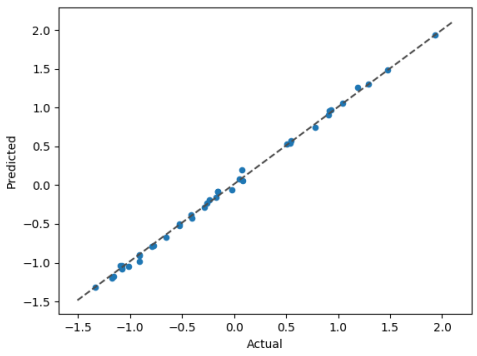


Figura 5.29: Scatter plot relativo a Convolution, dataset con tutti gli errori (input set 26), 20 layers

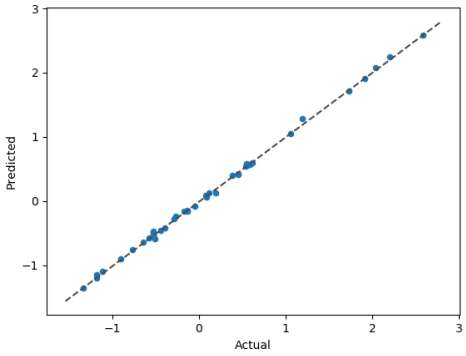


Figura 5.30: Scatter plot relativo a Convolution, dataset senza errori > 0.95 ed esteso per introdurre i vincoli del grafo (input set 26), 20 layers

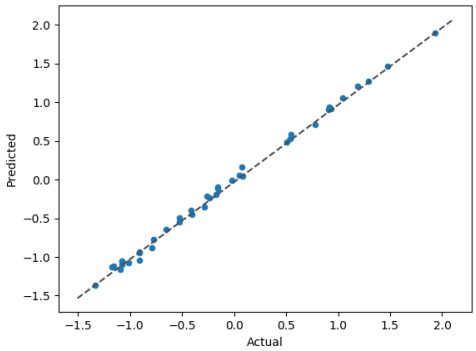


Figura 5.31: Scatter plot relativo a Convolution, dataset con tutti gli errori ed esteso per introdurre i vincoli del grafo (input set 26), 20 layers

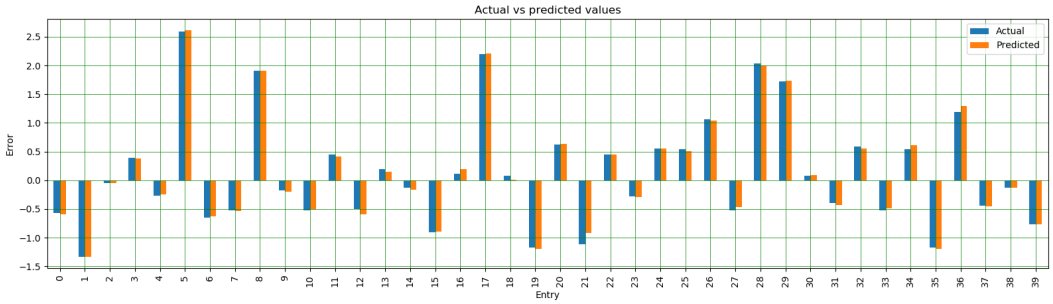


Figura 5.32: Confronto tra valori predetti e valori reali (Convolution, dataset senza errori > 0.95, input set 26, 20 layers)

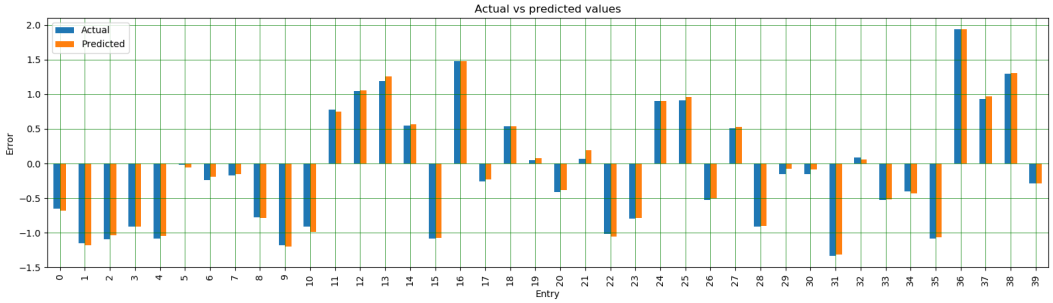


Figura 5.33: Confronto tra valori predetti e valori reali (Convolution, dataset con tutti gli errori, input set 26, 20 layers)

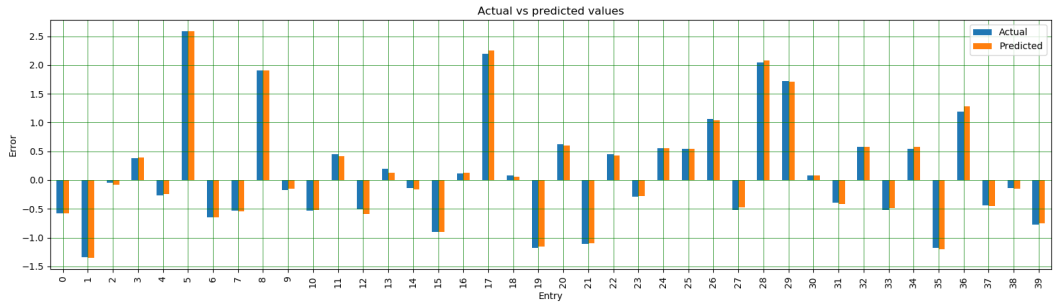


Figura 5.34: Confronto tra valori predetti e valori reali (Convolution, dataset senza errori > 0.95 ed esteso per introdurre i vincoli del grafo, input set 26, 20 layers)

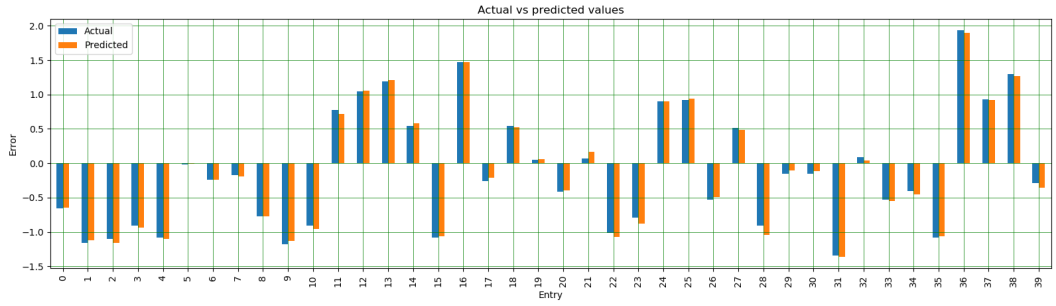


Figura 5.35: Confronto tra valori predetti e valori reali (Convolution, dataset con tutti gli errori ed esteso per introdurre i vincoli del grafo, input set 26, 20 layers)

CONVOLUTION	Mode	Dataset	Time to train
5 layers	All errors	Standard	28.08s
		Improved	29.75s
	Skip errors > 0.95	Standard	41.93s
		Improved	77.2s
20 layers	All errors	Standard	134.24s
		Improved	137.6s
	Skip errors > 0.95	Standard	238.44s
		Improved	286.7s

Figura 5.36: Tempi di training dei vari modelli di deep learning

Come è ben possibile osservare dalla figura 5.36, i tempi di training non sono per nulla trascurabili: anche se si parte da valori comunque comparabili con quelli ottenuti dal modello di regressione, all'aumentare della complessità, incrementata sia dal numero di esempi più alto considerando la modalità con tutti gli errori sia dall'introduzione di ulteriori variabili per rappresentare i vincoli del grafo, aumenta anche vertiginosamente il tempo richiesto per la computazione effettiva.

Deep learning results

(n=number of features)

Benchmark: convolution

IS6 Shape: 10n*10n*10n*10n*2 (5 layers)		IS6 (Improved) Shape: 10n*10n*10n*10n*2 (5 layers)		IS6 Shape: 100n*100n*100n*100n*2 (5 layers)		IS6 (Improved) Shape: 100n*100n*100n*100n*2 (5 layers)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 9366 std_RMSE: 0.04 std_MAE: 0.029 abs_MAE: 0.326 norm_MAE: 0.02 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.998	Total Entries: 9373 std_RMSE: 0.11 std_MAE: 0.032 abs_MAE: 0.366 norm_MAE: 0.022 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988	Total Entries: 9366 std_RMSE: 0.043 std_MAE: 0.029 abs_MAE: 0.327 norm_MAE: 0.02 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.998	Total Entries: 9373 std_RMSE: 0.11 std_MAE: 0.032 abs_MAE: 0.364 norm_MAE: 0.022 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988	Total Entries: 9366 std_RMSE: 0.034 std_MAE: 0.022 abs_MAE: 0.252 norm_MAE: 0.015 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.999	Total Entries: 9373 std_RMSE: 0.108 std_MAE: 0.026 abs_MAE: 0.298 norm_MAE: 0.018 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988	Total Entries: 9366 std_RMSE: 0.033 std_MAE: 0.022 abs_MAE: 0.244 norm_MAE: 0.015 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.999	Total Entries: 9373 std_RMSE: 0.108 std_MAE: 0.027 abs_MAE: 0.305 norm_MAE: 0.019 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988

IS6 Shape: 10n*10n*10n*10n*10n* 10n*10n*10n*10n*2 (10 layers)		IS6 (Improved) Shape: 10n*10n*10n*10n*10n* 10n*10n*10n*10n*2 (10 layers)		IS6 Shape: 4n*8n*20n*3n*2n*4n*6n* 2n*7n*2 (10 layers)		IS6 (Improved) Shape: 4n*8n*20n*3n*2n*4n*6n* 2n*7n*2 (10 layers)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 9366 std_RMSE: 0.037 std_MAE: 0.026 abs_MAE: 0.293 norm_MAE: 0.018 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.999	Total Entries: 9373 std_RMSE: 0.11 std_MAE: 0.033 abs_MAE: 0.379 norm_MAE: 0.023 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988	Total Entries: 9366 std_RMSE: 0.037 std_MAE: 0.026 abs_MAE: 0.292 norm_MAE: 0.018 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.999	Total Entries: 9373 std_RMSE: 0.109 std_MAE: 0.032 abs_MAE: 0.361 norm_MAE: 0.022 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988	Total Entries: 9366 std_RMSE: 0.05 std_MAE: 0.034 abs_MAE: 0.383 norm_MAE: 0.023 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.998	Total Entries: 9373 std_RMSE: 0.11 std_MAE: 0.033 abs_MAE: 0.373 norm_MAE: 0.023 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988	Total Entries: 9366 std_RMSE: 0.04 std_MAE: 0.029 abs_MAE: 0.326 norm_MAE: 0.02 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.998	Total Entries: 9373 std_RMSE: 0.11 std_MAE: 0.032 abs_MAE: 0.367 norm_MAE: 0.022 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988

IS6 Shape: 2n*2n*2n*2n*2n*2n*2n* 2n*2n*2n*2n*2n*2n*2n*2n* 2n*2n*2n*2n*2 (20 layers)		IS6 (Improved) Shape: 2n*2n*2n*2n*2n*2n*2n* 2n*2n*2n*2n*2n*2n*2n*2n* 2n*2n*2n*2n*2 (20 layers)		IS6 Shape: 10n*10n*10n*10n*10n* 10n*10n*10n*10n*10n*10n* 10n*10n*10n*10n*10n* 10n*10n*10n*2 (20 layers)		IS6 (Improved) Shape: 10n*10n*10n*10n*10n* 10n*10n*10n*10n*10n*10n* 10n*10n*10n*10n*10n* 10n*10n*10n*2 (20 layers)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 9366 std_RMSE: 0.052 std_MAE: 0.036 abs_MAE: 0.404 norm_MAE: 0.024 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.997	Total Entries: 9373 std_RMSE: 0.119 std_MAE: 0.042 abs_MAE: 0.472 norm_MAE: 0.029 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.986	Total Entries: 9366 std_RMSE: 0.047 std_MAE: 0.033 abs_MAE: 0.38 norm_MAE: 0.023 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.998	Total Entries: 9373 std_RMSE: 0.116 std_MAE: 0.043 abs_MAE: 0.486 norm_MAE: 0.029 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.986	Total Entries: 9366 std_RMSE: 0.038 std_MAE: 0.026 abs_MAE: 0.298 norm_MAE: 0.018 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.999	Total Entries: 9373 std_RMSE: 0.111 std_MAE: 0.034 abs_MAE: 0.384 norm_MAE: 0.023 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988	Total Entries: 9366 std_RMSE: 0.034 std_MAE: 0.024 abs_MAE: 0.274 norm_MAE: 0.017 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.999	Total Entries: 9373 std_RMSE: 0.109 std_MAE: 0.029 abs_MAE: 0.334 norm_MAE: 0.02 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988

IS6 Shape: 5n*9n*15n*4n*7n*6n*3n* 8n*85n*47n*32n*4n*9n* 21n*2n*14n*36n*5n*7n*2 (20 layers)		IS6 (Improved) Shape: 5n*9n*15n*4n*7n*6n*3n* 8n*85n*47n*32n*4n*9n* 21n*2n*14n*36n*5n*7n*2 (20 layers)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 9366 std_RMSE: 0.048 std_MAE: 0.034 abs_MAE: 0.39 norm_MAE: 0.024 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.998	Total Entries: 9373 std_RMSE: 0.111 std_MAE: 0.034 abs_MAE: 0.385 norm_MAE: 0.023 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988	Total Entries: 9366 std_RMSE: 0.038 std_MAE: 0.026 abs_MAE: 0.296 norm_MAE: 0.018 abs_mean: 16.509 abs_var: 128.639 R2_score: 0.999	Total Entries: 9373 std_RMSE: 0.108 std_MAE: 0.027 abs_MAE: 0.305 norm_MAE: 0.018 abs_mean: 16.491 abs_var: 128.952 R2_score: 0.988

Figura 5.37: Confronto tra gli indici di qualità nei vari modelli di deep learning

Invece, nelle figure sottostanti vengono riportati solamente due grafici per quanto riguarda il benchmark di Jacobi, poiché gli andamenti di tutti i modelli di deep learning sono abbastanza simili tra loro.

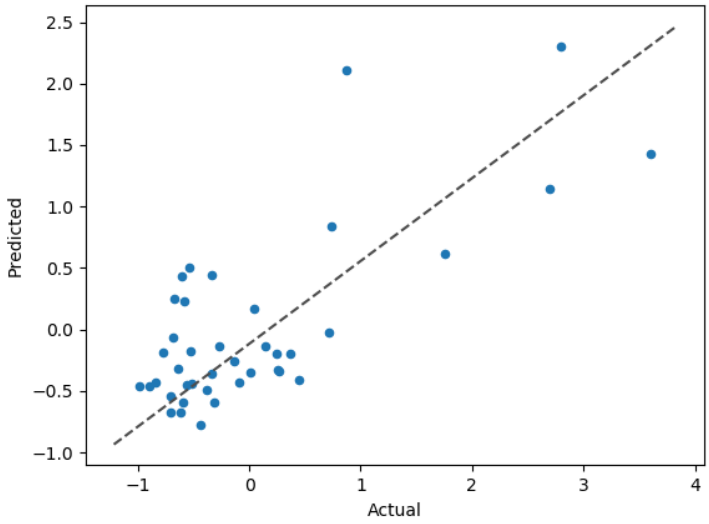


Figura 5.38: Scatter plot relativo a Jacobi, dataset con tutti gli errori ed esteso per introdurre i vincoli del grafo (input set 12), 10 layers

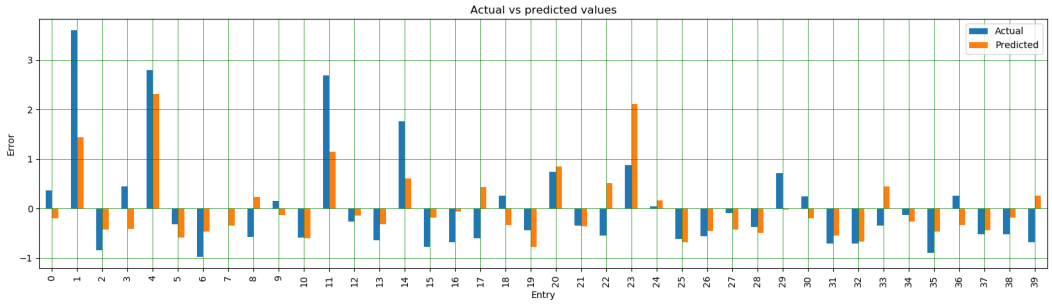


Figura 5.39: Confronto tra valori predetti e valori reali (Jacobi, dataset senza errori > 0.95 ed esteso per introdurre i vincoli del grafo, input set 15, 10 layers)

Deep learning results

(n=number of features)

Benchmark: Jacobi

IS29 Shape: 10n*10n*10n*10n^2 (5 layers)		IS29 (Improved) Shape: 10n*10n*10n*10n^2 (5 layers)		IS29 Shape: 100n*100n*100n*100n^2 (5 layers)		IS29 (Improved) Shape: 100n*100n*100n*100n^2 (5 layers)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 252 std_RMSE: 1.106 std_MAE: 0.9 abs_MAE: 3.617 norm_MAE: 0.495 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.081	Total Entries: 3645 std_RMSE: 0.721 std_MAE: 0.512 abs_MAE: 3.135 norm_MAE: -0.312 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.529	Total Entries: 252 std_RMSE: 1.26 std_MAE: 1.033 abs_MAE: 4.151 norm_MAE: 0.568 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.404	Total Entries: 3645 std_RMSE: 0.629 std_MAE: 0.432 abs_MAE: 2.646 norm_MAE: -0.263 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.642	Total Entries: 252 std_RMSE: 1.196 std_MAE: 1.033 abs_MAE: 4.153 norm_MAE: 0.568 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.263	Total Entries: 3645 std_RMSE: 0.699 std_MAE: 0.487 abs_MAE: 2.979 norm_MAE: -0.296 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.558	Total Entries: 252 std_RMSE: 1.369 std_MAE: 1.148 abs_MAE: 4.616 norm_MAE: 0.631 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.656	Total Entries: 3645 std_RMSE: 0.678 std_MAE: 0.478 abs_MAE: 2.925 norm_MAE: -0.291 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.584

IS29 Shape: 10n*10n*10n*10n*10n* 10n*10n*10n*10n^2 (10 layers)		IS29 (Improved) Shape: 10n*10n*10n*10n*10n* 10n*10n*10n*10n^2 (10 layers)		IS29 Shape: 4n*8n*20n*3n*2n*4n*6n* 2n*7n^2 (10 layers)		IS29 (Improved) Shape: 4n*8n*20n*3n*2n*4n*6n* 2n*7n^2 (10 layers)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 252 std_RMSE: 1.237 std_MAE: 1.038 abs_MAE: 4.171 norm_MAE: 0.57 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.351	Total Entries: 3645 std_RMSE: 0.732 std_MAE: 0.507 abs_MAE: 3.101 norm_MAE: -0.309 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.516	Total Entries: 252 std_RMSE: 1.272 std_MAE: 1.047 abs_MAE: 4.209 norm_MAE: 0.575 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.43	Total Entries: 3645 std_RMSE: 0.641 std_MAE: 0.438 abs_MAE: 2.682 norm_MAE: -0.267 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.629	Total Entries: 252 std_RMSE: 1.235 std_MAE: 1.02 abs_MAE: 4.1 norm_MAE: 0.561 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.348	Total Entries: 3645 std_RMSE: 0.76 std_MAE: 0.529 abs_MAE: 3.24 norm_MAE: -0.323 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.477	Total Entries: 252 std_RMSE: 1.151 std_MAE: 0.928 abs_MAE: 3.732 norm_MAE: 0.51 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.171	Total Entries: 3645 std_RMSE: 0.651 std_MAE: 0.459 abs_MAE: 2.812 norm_MAE: -0.28 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.617

IS29 Shape: 2n*2n*2n*2n*2n*2n*2n* 2n*2n*2n*2n*2n*2n*2n* 2n*2n*2n*2n^2 (20 layers)		IS29 (Improved) Shape: 2n*2n*2n*2n*2n*2n*2n* 2n*2n*2n*2n*2n*2n*2n* 2n*2n*2n*2n^2 (20 layers)		IS29 Shape: 10n*10n*10n*10n*10n* 10n*10n*10n*10n*10n*10n* 10n*10n*10n*10n*10n* 10n*10n*10n^2 (20 layers)		IS29 (Improved) Shape: 10n*10n*10n*10n*10n* 10n*10n*10n*10n*10n*10n* 10n*10n*10n*10n*10n* 10n*10n*10n^2 (20 layers)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 252 std_RMSE: 1.067 std_MAE: 0.849 abs_MAE: 3.414 norm_MAE: 0.467 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.006	Total Entries: 3645 std_RMSE: 0.795 std_MAE: 0.568 abs_MAE: 3.476 norm_MAE: -0.346 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.429	Total Entries: 252 std_RMSE: 1.235 std_MAE: 0.989 abs_MAE: 3.975 norm_MAE: 0.543 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.348	Total Entries: 3645 std_RMSE: 0.727 std_MAE: 0.519 abs_MAE: 3.176 norm_MAE: -0.316 abs_mean: -10.047 abs_var: 37.467 R2_score: 0.522	Total Entries: 252 std_RMSE: 1.069 std_MAE: 0.85 abs_MAE: 3.419 norm_MAE: 0.467 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.01	Total Entries: 3645 std_RMSE: 1.054 std_MAE: 0.752 abs_MAE: 4.604 norm_MAE: -0.458 abs_mean: -10.047 abs_var: 37.467 R2_score: -0.004	Total Entries: 252 std_RMSE: 1.07 std_MAE: 0.851 abs_MAE: 3.42 norm_MAE: 0.468 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.011	Total Entries: 3645 std_RMSE: 1.054 std_MAE: 0.752 abs_MAE: 4.604 norm_MAE: -0.458 abs_mean: -10.047 abs_var: 37.467 R2_score: -0.004

IS29 Shape: 5n*9n*15n*4n*7n*6n*3n* 8n*85n*47n*32n*4n*9n* 21n*2n*14n*36n*5n*7n*2 (20 layers)		IS29 (Improved) Shape: 5n*9n*15n*4n*7n*6n*3n* 8n*85n*47n*32n*4n*9n* 21n*2n*14n*36n*5n*7n*2 (20 layers)	
Skip errors > 0.95	**All errors**	**Skip errors > 0.95**	**All errors**
Total Entries: 252 std_RMSE: 1.068 std_MAE: 0.849 abs_MAE: 3.415 norm_MAE: 0.467 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.008	Total Entries: 3645 std_RMSE: 1.054 std_MAE: 0.752 abs_MAE: 4.604 norm_MAE: -0.458 abs_mean: -10.047 abs_var: 37.467 R2_score: -0.004	Total Entries: 252 std_RMSE: 1.07 std_MAE: 0.851 abs_MAE: 3.42 norm_MAE: 0.468 abs_mean: 7.315 abs_var: 16.162 R2_score: -0.011	Total Entries: 3645 std_RMSE: 1.054 std_MAE: 0.752 abs_MAE: 4.601 norm_MAE: -0.458 abs_mean: -10.047 abs_var: 37.467 R2_score: -0.004

Figura 5.40: Confronto tra gli indici di qualità nei vari modelli di deep learning

In definitiva, come si può notare dai risultati qui presentati, l'approccio del deep learning è abbastanza preciso e riesce effettivamente a produrre ottimi indici di qualità per quanto riguarda alcuni benchmark come Convolution o DWT. Risulta ancora però lacunoso se si considerano quei dataset in cui si ha una grande quantità di variabili: infatti il deep learning in Jacobi e Blackscholes non funziona benissimo, in quanto i risultati ottenuti sono abbastanza scarsi.

Inoltre, proprio per il fatto che le features sono tante, i tempi computazionali per il training dei vari modelli aumentano drasticamente.
Infine, espandendo il dataset con features aggiuntive introdotte dai vincoli dati dal grafo, fa in modo di ottenere in tutti i casi un miglioramento: meno evidente in quei benchmark che già danno di per sè degli ottimi risultati anche con un dataset di dimensioni standard, ma più accentuato ad esempio in BlackScholes e Jacobi, dove gli approcci classici non riescono a dare ottimi risultati finali.

5.3 GCN

In questo sottocapitolo vengono riportati i risultati sperimentali per quanto riguarda i datasets del benchmark DWT.

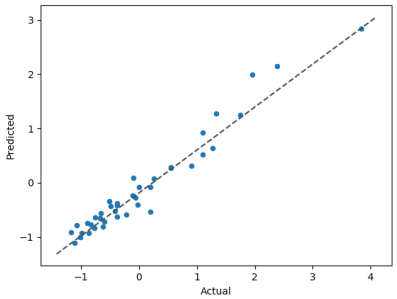


Figura 5.41: Scatter plot relativo a DWT, dataset senza errori > 0.95 (input set 1)

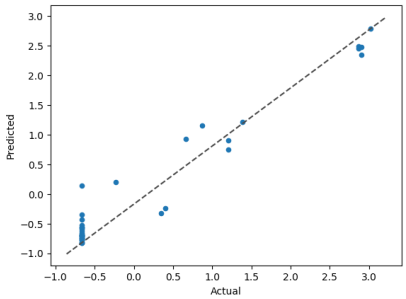


Figura 5.42: Scatter plot relativo a DWT, dataset con tutti gli errori (input set 1)

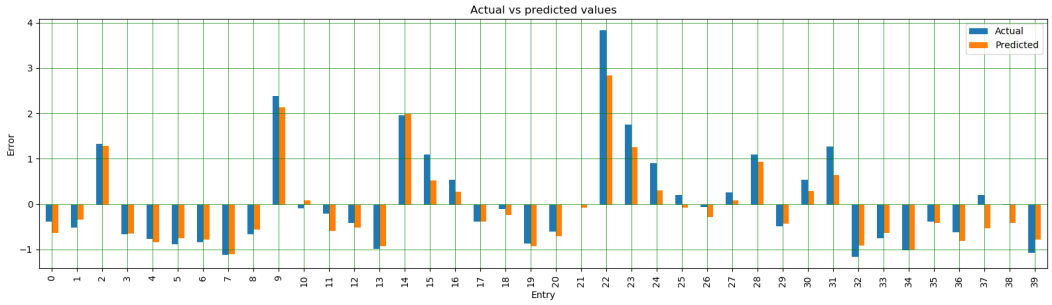


Figura 5.43: Confronto tra valori predetti e valori reali (DWT, dataset senza errori > 0.95, input set 1)

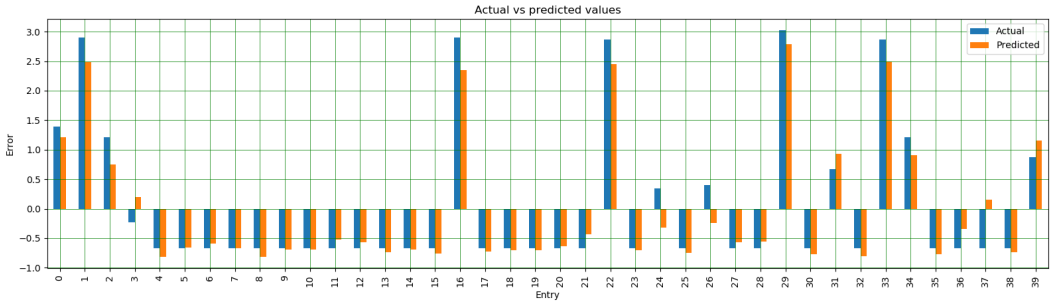


Figura 5.44: Confronto tra valori predetti e valori reali (DWT, dataset con tutti gli errori, input set 1)

Come evidenziato dai grafici qua riportati, questo approccio nuovo non restituisce scarsi risultati considerando DWT e perciò sembrerebbe una valida alternativa agli approcci basati sull'inserimento di features aggiuntive nel training set.
Purtroppo, come verrà spiegato meglio nel prossimo sottocapitolo e come già evidenziato dalla figura sottostante, ciò non è corretto: in alcuni datasets relativi a certi benchmark più complessi, l'approccio risulta infatti ancora acerbo e modelli più tradizionali come quelli di deep learning, performano nettamente meglio.

GCN results

	IS2	
Benchmark	**Skip errors > 0.95**	**All errors**
convolution	Total Entries: 9367 std_RMSE: 0.051 std_MAE: 0.032 abs_MAE: 0.368 norm_MAE: 0.022 abs_mean: 16.506 abs_var: 128.382 R2_score: 0.997	Total Entries: 9373 std_RMSE: 0.06 std_MAE: 0.026 abs_MAE: 0.3 norm_MAE: 0.018 abs_mean: 16.491 abs_var: 128.651 R2_score: 0.996
correlation	Total Entries: 6149 std_RMSE: 0.633 std_MAE: 0.492 abs_MAE: 4.557 norm_MAE: 0.304 abs_mean: 14.975 abs_var: 85.669 R2_score: 0.582	Total Entries: 9977 std_RMSE: 0.805 std_MAE: 0.65 abs_MAE: 10.515 norm_MAE: 2.288 abs_mean: 4.597 abs_var: 261.662 R2_score: 0.349

dwt	Total Entries: 2551 std_RMSE: 0.442 std_MAE: 0.349 abs_MAE: 2.298 norm_MAE: 0.249 abs_mean: 9.21 abs_var: 43.235 R2_score: 0.82	Total Entries: 10000 std_RMSE: 0.212 std_MAE: 0.12 abs_MAE: 0.931 norm_MAE: -0.35 abs_mean: -2.656 abs_var: 59.902 R2_score: 0.957
BlackScholes	Total Entries: 3354 std_RMSE: 0.931 std_MAE: 0.748 abs_MAE: 3.357 norm_MAE: 0.668 abs_mean: 5.028 abs_var: 20.152 R2_score: 0.058	Total Entries: 10000 std_RMSE: 0.905 std_MAE: 0.674 abs_MAE: 4.752 norm_MAE: -2.148 abs_mean: -2.212 abs_var: 49.678 R2_score: 0.17
Jacobi	Total Entries: 274 std_RMSE: 1.143 std_MAE: 0.844 abs_MAE: 3.655 norm_MAE: 0.49 abs_mean: 7.456 abs_var: 18.737 R2_score: 0.011	Total Entries: 3645 std_RMSE: 0.902 std_MAE: 0.621 abs_MAE: 3.898 norm_MAE: -0.395 abs_mean: -9.871 abs_var: 39.418 R2_score: 0.289

Figura 5.45: Confronto tra gli indici di qualità relativi alle GCN per tutti i benchmark sull'input set 2

5.4 Confronto tra i vari modelli

Tutti gli esperimenti effettuati utilizzando i datasets generati a partire da ogni benchmark, hanno portato a risultati interessanti, sia positivi che negativi e quindi sia indicanti modelli di machine learning per nulla adatti a risolvere il problema del dominio applicativo, sia segnalanti invece approcci molto più sofisticati e precisi.
Di seguito, si riporta una tabella comparativa in cui sono presentati tutti gli indici di qualità calcolati come media di tutti i valori su tutti gli input set. Nel caso dei modelli di deep learning, questa media è stata applicata anche a tutte le varie tipologie studiate.

Results comparator
(avg indexes values for each model)

	Regressor		Regressor (Improved)		Deep learning		Deep learning (Improved)		GCN	
Benchmark	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**	**Skip errors > 0.95**	**All errors**
convolution	Total Entries: 9366 std_RMSE: 0.698 std_MAE: 0.563 abs_MAE: 6.38 norm_MAE: 0.386 abs_mean: 16.505 abs_var: 128.458 R2_score: 0.518	Total Entries: 9373 std_RMSE: 0.696 std_MAE: 0.559 abs_MAE: 6.347 norm_MAE: 0.385 abs_mean: 16.489 abs_var: 128.736 R2_score: 0.509	Total Entries: 9366 std_RMSE: 0.698 std_MAE: 0.563 abs_MAE: 6.379 norm_MAE: 0.386 abs_mean: 16.505 abs_var: 128.458 R2_score: 0.518	Total Entries: 9373 std_RMSE: 0.696 std_MAE: 0.56 abs_MAE: 6.354 norm_MAE: 0.385 abs_mean: 16.489 abs_var: 128.736 R2_score: 0.509	Total Entries: 9366 std_RMSE: 0.057 std_MAE: 0.041 abs_MAE: 0.464 norm_MAE: 0.028 abs_mean: 16.505 abs_var: 128.458 R2_score: 0.983	Total Entries: 9373 std_RMSE: 0.132 std_MAE: 0.055 abs_MAE: 0.624 norm_MAE: 0.038 abs_mean: 16.489 abs_var: 128.736 R2_score: 0.96	Total Entries: 9366 std_RMSE: 0.038 std_MAE: 0.027 abs_MAE: 0.302 norm_MAE: 0.018 abs_mean: 16.505 abs_var: 128.458 R2_score: 0.999	Total Entries: 9373 std_RMSE: 0.105 std_MAE: 0.031 abs_MAE: 0.354 norm_MAE: 0.021 abs_mean: 16.489 abs_var: 128.736 R2_score: 0.989	Total Entries: 9366 std_RMSE: 0.053 std_MAE: 0.038 abs_MAE: 0.427 norm_MAE: 0.026 abs_mean: 16.505 abs_var: 128.458 R2_score: 0.997	Total Entries: 9373 std_RMSE: 0.112 std_MAE: 0.043 abs_MAE: 0.487 norm_MAE: 0.03 abs_mean: 16.489 abs_var: 128.736 R2_score: 0.987
correlation	Total Entries: 5854 std_RMSE: 0.769 std_MAE: 0.622 abs_MAE: 5.804 norm_MAE: 0.381 abs_mean: 15.258 abs_var: 87.339 R2_score: 0.405	Total Entries: 9977 std_RMSE: 0.85 std_MAE: 0.697 abs_MAE: 10.973 norm_MAE: 3.076 abs_mean: 4.474 abs_var: 247.75 R2_score: 0.283	Total Entries: 5854 std_RMSE: 0.769 std_MAE: 0.622 abs_MAE: 5.808 norm_MAE: 0.381 abs_mean: 15.258 abs_var: 87.339 R2_score: 0.404	Total Entries: 9977 std_RMSE: 0.85 std_MAE: 0.698 abs_MAE: 10.98 norm_MAE: 3.078 abs_mean: 4.474 abs_var: 247.75 R2_score: 0.283	Total Entries: 5854 std_RMSE: 0.272 std_MAE: 0.179 abs_MAE: 1.678 norm_MAE: 0.11 abs_mean: 15.258 abs_var: 87.339 R2_score: 0.924	Total Entries: 9977 std_RMSE: 0.36 std_MAE: 0.239 abs_MAE: 3.77 norm_MAE: 1.12 abs_mean: 4.474 abs_var: 247.75 R2_score: 0.867	Total Entries: 5854 std_RMSE: 0.254 std_MAE: 0.157 abs_MAE: 1.47 norm_MAE: 0.096 abs_mean: 15.258 abs_var: 87.339 R2_score: 0.935	Total Entries: 9977 std_RMSE: 0.335 std_MAE: 0.208 abs_MAE: 3.284 norm_MAE: 0.985 abs_mean: 4.474 abs_var: 247.75 R2_score: 0.887	Total Entries: 5854 std_RMSE: 0.634 std_MAE: 0.496 abs_MAE: 4.63 norm_MAE: 0.304 abs_mean: 15.258 abs_var: 87.339 R2_score: 0.594	Total Entries: 9977 std_RMSE: 0.809 std_MAE: 0.641 abs_MAE: 10.079 norm_MAE: 2.817 abs_mean: 4.474 abs_var: 247.75 R2_score: 0.35
dwt	Total Entries: 3075 std_RMSE: 0.813 std_MAE: 0.652 abs_MAE: 4.327 norm_MAE: 0.49 abs_mean: 8.837 abs_var: 44.211 R2_score: 0.347	Total Entries: 10000 std_RMSE: 0.807 std_MAE: 0.634 abs_MAE: 5.107 norm_MAE: 3.876 abs_mean: 1.96 abs_var: 65.258 R2_score: 0.376	Total Entries: 3075 std_RMSE: 0.814 std_MAE: 0.652 abs_MAE: 4.327 norm_MAE: 0.49 abs_mean: 8.837 abs_var: 44.211 R2_score: 0.347	Total Entries: 10000 std_RMSE: 0.808 std_MAE: 0.635 abs_MAE: 5.108 norm_MAE: 3.877 abs_mean: 1.96 abs_var: 65.258 R2_score: 0.376	Total Entries: 3075 std_RMSE: 0.249 std_MAE: 0.191 abs_MAE: 1.256 norm_MAE: 0.142 abs_mean: 8.837 abs_var: 44.211 R2_score: 0.932	Total Entries: 10000 std_RMSE: 0.17 std_MAE: 0.076 abs_MAE: 0.608 norm_MAE: 0.446 abs_mean: 1.96 abs_var: 65.258 R2_score: 0.971	Total Entries: 3075 std_RMSE: 0.223 std_MAE: 0.171 abs_MAE: 1.125 norm_MAE: 0.127 abs_mean: 8.837 abs_var: 44.211 R2_score: 0.947	Total Entries: 10000 std_RMSE: 0.156 std_MAE: 0.068 abs_MAE: 0.542 norm_MAE: 0.395 abs_mean: 1.96 abs_var: 65.258 R2_score: 0.976	Total Entries: 3075 std_RMSE: 0.281 std_MAE: 0.215 abs_MAE: 1.419 norm_MAE: 0.161 abs_mean: 8.837 abs_var: 44.211 R2_score: 0.919	Total Entries: 10000 std_RMSE: 0.184 std_MAE: 0.099 abs_MAE: 0.795 norm_MAE: 0.587 abs_mean: 1.96 abs_var: 65.258 R2_score: 0.967
Jacobi	Total Entries: 245 std_RMSE: 1.09 std_MAE: 0.863 abs_MAE: 3.498 norm_MAE: 0.485 abs_mean: 7.2 abs_var: 16.383 R2_score: 0.122	Total Entries: 3645 std_RMSE: 0.937 std_MAE: 0.679 abs_MAE: 4.104 norm_MAE: 0.406 abs_mean: 10.141 abs_var: 36.599 R2_score: 0.233	Total Entries: 245 std_RMSE: 1.092 std_MAE: 0.865 abs_MAE: 3.505 norm_MAE: 0.486 abs_mean: 7.2 abs_var: 16.383 R2_score: 0.125	Total Entries: 3645 std_RMSE: 0.938 std_MAE: 0.679 abs_MAE: 4.106 norm_MAE: 0.406 abs_mean: 10.141 abs_var: 36.599 R2_score: 0.232	Total Entries: 245 std_RMSE: 1.065 std_MAE: 0.83 abs_MAE: 3.363 norm_MAE: 0.466 abs_mean: 7.2 abs_var: 16.383 R2_score: 0.091	Total Entries: 3645 std_RMSE: 0.814 std_MAE: 0.572 abs_MAE: 3.455 norm_MAE: 0.342 abs_mean: 10.141 abs_var: 36.599 R2_score: 0.405	Total Entries: 245 std_RMSE: 1.087 std_MAE: 0.846 abs_MAE: 3.428 norm_MAE: 0.475 abs_mean: 7.2 abs_var: 16.383 R2_score: 0.123	Total Entries: 3645 std_RMSE: 0.811 std_MAE: 0.568 abs_MAE: 3.432 norm_MAE: 0.34 abs_mean: 10.141 abs_var: 36.599 R2_score: 0.396	Total Entries: 245 std_RMSE: 1.068 std_MAE: 0.833 abs_MAE: 3.375 norm_MAE: 0.467 abs_mean: 7.2 abs_var: 16.383 R2_score: 0.12	Total Entries: 3645 std_RMSE: 0.876 std_MAE: 0.627 abs_MAE: 3.789 norm_MAE: 0.375 abs_mean: 10.141 abs_var: 36.599 R2_score: 0.329
BlackScholes	Total Entries: 3309 std_RMSE: 0.893 std_MAE: 0.707 abs_MAE: 3.152 norm_MAE: 0.621 abs_mean: 5.082 abs_var: 19.875 R2_score: 0.175	Total Entries: 10000 std_RMSE: 0.887 std_MAE: 0.681 abs_MAE: 4.557 norm_MAE: 2.126 abs_mean: 2.178 abs_var: 44.78 R2_score: 0.198	Total Entries: 3309 std_RMSE: 0.896 std_MAE: 0.71 abs_MAE: 3.164 norm_MAE: 0.623 abs_mean: 5.082 abs_var: 19.875 R2_score: 0.17	Total Entries: 10000 std_RMSE: 0.886 std_MAE: 0.681 abs_MAE: 4.555 norm_MAE: 2.125 abs_mean: 2.178 abs_var: 44.78 R2_score: 0.199	Total Entries: 3309 std_RMSE: 0.742 std_MAE: 0.563 abs_MAE: 2.508 norm_MAE: 0.494 abs_mean: 5.082 abs_var: 19.874 R2_score: 0.416	Total Entries: 10000 std_RMSE: 0.568 std_MAE: 0.431 abs_MAE: 2.879 norm_MAE: 1.345 abs_mean: 2.178 abs_var: 44.78 R2_score: 0.652	Total Entries: 3309 std_RMSE: 0.744 std_MAE: 0.567 abs_MAE: 2.526 norm_MAE: 0.497 abs_mean: 5.082 abs_var: 19.874 R2_score: 0.398	Total Entries: 10000 std_RMSE: 0.621 std_MAE: 0.462 abs_MAE: 3.091 norm_MAE: 1.443 abs_mean: 2.178 abs_var: 44.78 R2_score: 0.544	Total Entries: 3309 std_RMSE: 0.943 std_MAE: 0.733 abs_MAE: 3.269 norm_MAE: 0.643 abs_mean: 5.082 abs_var: 19.875 R2_score: 0.082	Total Entries: 10000 std_RMSE: 0.891 std_MAE: 0.686 abs_MAE: 4.588 norm_MAE: 2.141 abs_mean: 2.178 abs_var: 44.78 R2_score: 0.191

Figura 5.46: Confronto tra gli indici di qualità relativi a tutti i modelli di machine learning testati

Infine, vengono riportati gli stessi valori in maniera più dettagliata considerando i singoli benchmark e confrontando gli indici al fine di verificare quale approccio di machine learning si sia rivelato più efficace sul dataset specifico.
Sull'asse x vengono indicati i modelli, che per ragioni di spazio vengono riassunti con un intero che ne rappresenta il tipo. In particolare, la legenda è la seguente:

1. Linear regressor, dataset normale;
2. Linear regressor, dataset esteso;
3. Deep learning, dataset normale, shape: $10n * 10n * 10n * 10n * 2$ (5 livelli);
4. Deep learning, dataset normale, shape: $100n * 100n * 100n * 100n * 2$ (5 livelli);
5. Deep learning, dataset normale, shape: $10n * 10n * 10n * 10n * 10n * 10n * 10n * 10n * 10n * 10n * 2$ (10 livelli);
6. Deep learning, dataset normale, shape: $4n * 8n * 20n * 3n * 2n * 4n * 6n * 2n * 7n * 2$ (10 livelli);

7. Deep learning, dataset normale, shape: $2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2$ (20 livelli);

8. Deep learning, dataset normale, shape: $10n * 2$ (20 livelli);

9. Deep learning, dataset normale, shape: $5n * 9n * 15n * 4n * 7n * 6n * 3n * 8n * 85n * 47n * 32n * 4n * 9n * 21n * 2n * 14n * 36n * 5n * 7n * 2$ (20 livelli);

10. Deep learning, dataset esteso, shape: $10n * 10n * 10n * 10n * 2$ (5 livelli);

11. Deep learning, dataset esteso, shape: $100n * 100n * 100n * 100n * 2$ (5 livelli);

12. Deep learning, dataset esteso, shape: $10n * 10n * 10n * 10n * 10n * 10n * 10n * 10n * 10n * 10n * 2$ (10 livelli);

13. Deep learning, dataset esteso, shape: $4n * 8n * 20n * 3n * 2n * 4n * 6n * 2n * 7n * 2$ (10 livelli);

14. Deep learning, dataset esteso, shape: $2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2n * 2$ (20 livelli);

15. Deep learning, dataset esteso, shape: $10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*10n*2$ (20 livelli);

16. Deep learning, dataset esteso, shape: $5n * 9n * 15n * 4n * 7n * 6n * 3n * 8n * 85n * 47n * 32n * 4n * 9n * 21n * 2n * 14n * 36n * 5n * 7n * 2$ (20 livelli);

17. GCN, dataset normale.

É opportuno precisare che in tutti i casi di deep learning, per n si intende il numero delle features.

Iniziando col presentare i dati relativi a Convolution, si può notare che questo benchmark, avendo di per sè un numero esiguo di variabili (4), non rappresenta un grosso problema per i vari modelli di machine learning: infatti, escludendo il modello di regressione lineare, approccio non

corretto in tutti i casi analizzati, gli indici di qualità presentano valori molto buoni con un R2_score che tocca quasi il valore perfetto, ovvero 1, indice di totale uguaglianza tra i valori reali e i valori predetti dal modello.
L'RMSE e il MAE presentano anch'essi valori molto bassi e anche l'approccio GCN non è da escludere per risolvere il problema (anche se i modelli di deep learning sono comunque migliori).

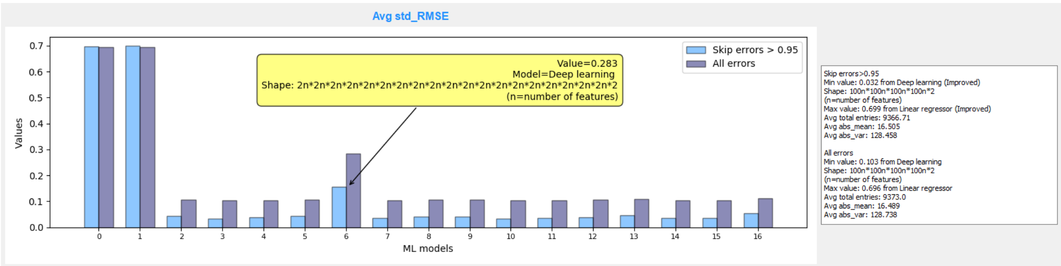


Figura 5.47: Confronto tra i valori di RMSE (Convolution)

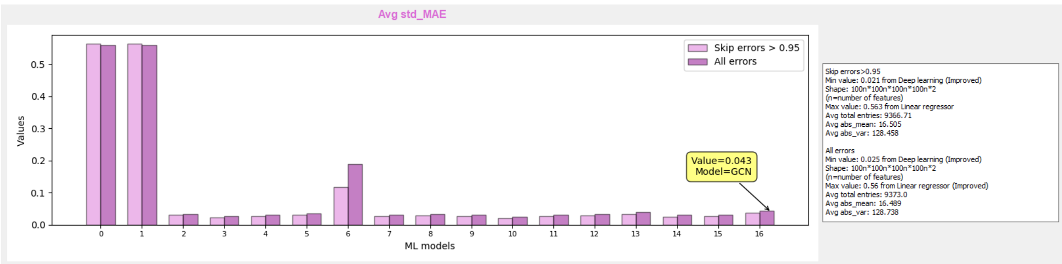


Figura 5.48: Confronto tra i valori di MAE (Convolution)

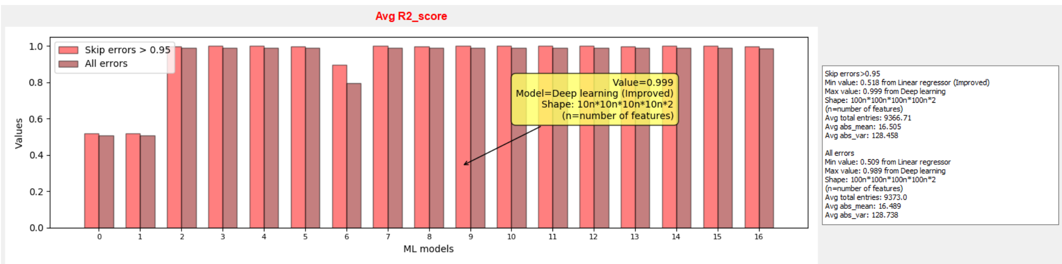


Figura 5.49: Confronto tra i valori di R2_score (Convolution)

Considerando Correlation e DWT (7 variabili), si può affermare che il primo presenta valori peggiori se confrontato al secondo e a Convolution, ed escludendo il modello di regressione lineare, come al solito del tutto

impreciso, si conferma come approccio vincente il deep learning, che raggiunge un R2_score di 0.94 nel modello con forma $100n * 100n * 100n * 100n * 2$ nel caso di dataset filtrato escludendo errori molto grandi e di 0.90 sempre con lo stessa topologia di rete considerando però il dataset completo.
Il discorso non è molto diverso per DWT che però presenta valori in generale migliori.

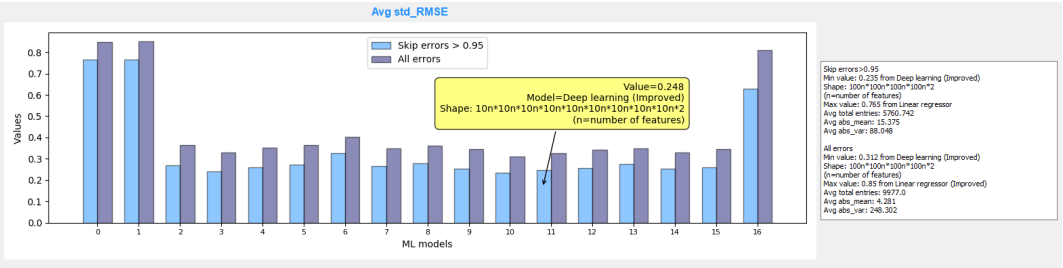


Figura 5.50: Confronto tra i valori di RMSE (Correlation)

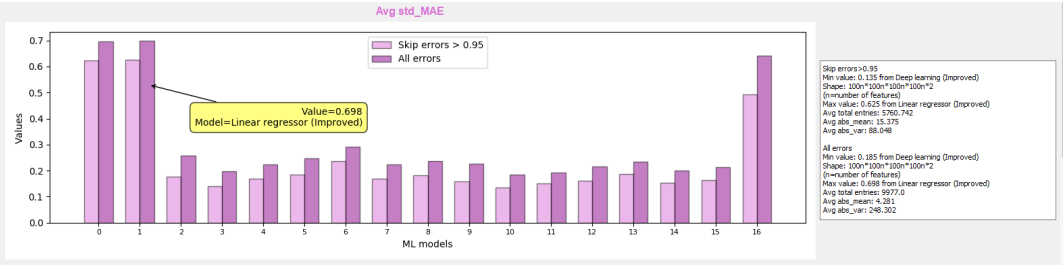


Figura 5.51: Confronto tra i valori di MAE (Correlation)

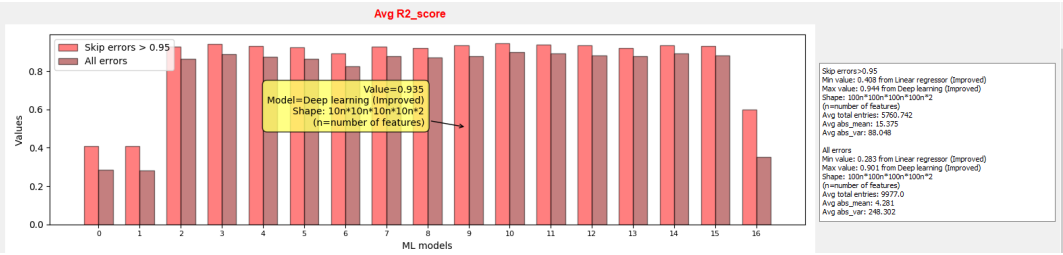


Figura 5.52: Confronto tra i valori di R2_score (Correlation)

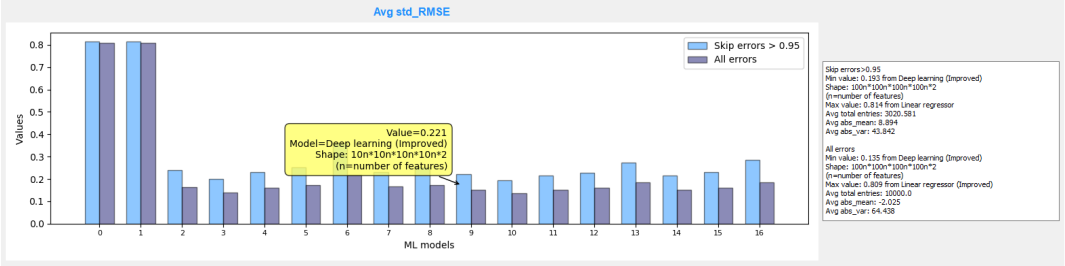


Figura 5.53: Confronto tra i valori di RMSE (DWT)

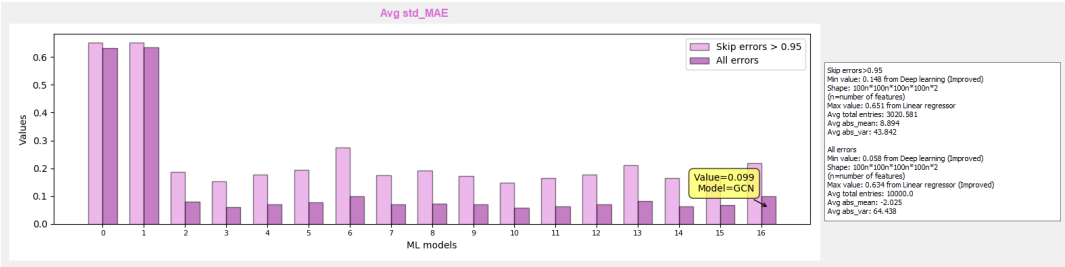


Figura 5.54: Confronto tra i valori di MAE (DWT)

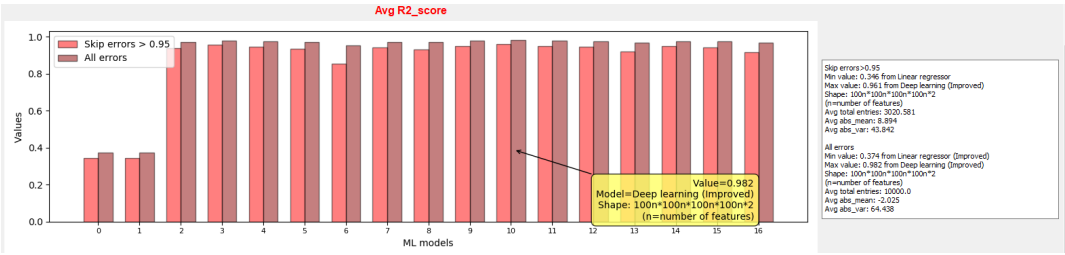


Figura 5.55: Confronto tra i valori di R2_score (DWT)

Aumentando ulteriormente la cardinalità delle variabili e quindi considerando prima BlackScholes (15) e poi Jacobi (25), il discorso però cambia drasticamente: i legami tra le variabili sono molto più complessi e articolati e perciò la funzione da imparare risulta ancora più difficile.
La diretta conseguenza di ciò è che in pratica tutti i modelli studiati non riescono ad approssimarla correttamente: il deep learning arriva a toccare valori di circa 0.90 di R2_score per BlackScholes ma con modelli molto profondi e che richiedono elevate capacità computazionali e tempi di calcolo. Per Jacobi inoltre anche gli approcci più complessi risultano del tutto inadeguati poiché i valori di RMSE e MAE toccano valori altissimi e l'R2_score valori bassissimi.

In entrambi i casi gli approcci di GCN falliscono immediatamente, poiché restituiscono risultati insoddisfacenti e in pochissimo tempo di training (di fatto quindi le reti non imparano affatto).

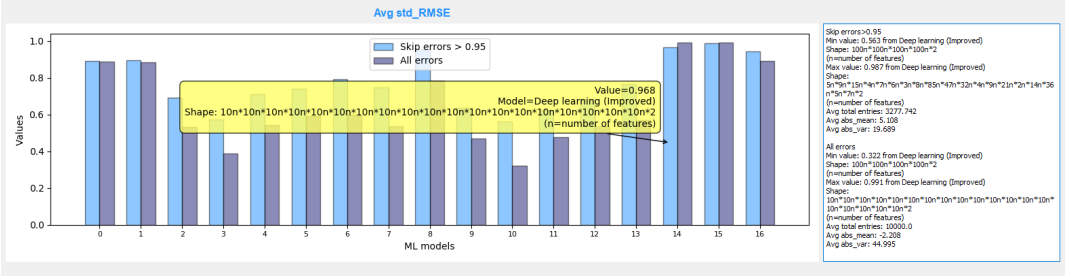


Figura 5.56: Confronto tra i valori di RMSE (BlackScholes)

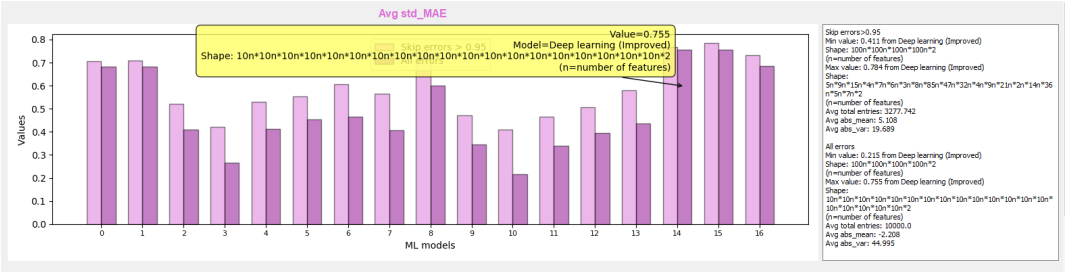


Figura 5.57: Confronto tra i valori di MAE (BlackScholes)

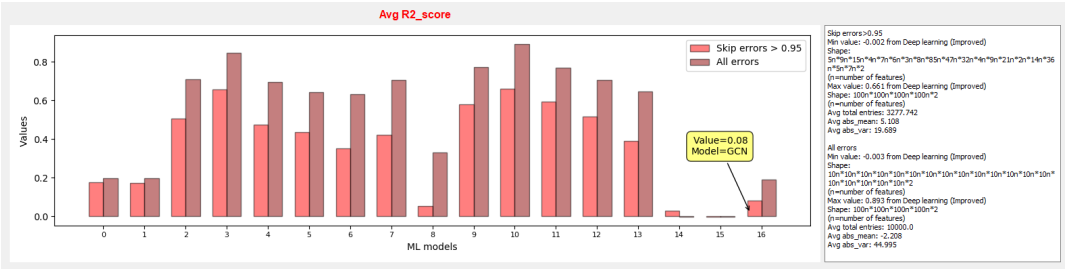


Figura 5.58: Confronto tra i valori di R2_score (BlackScholes)

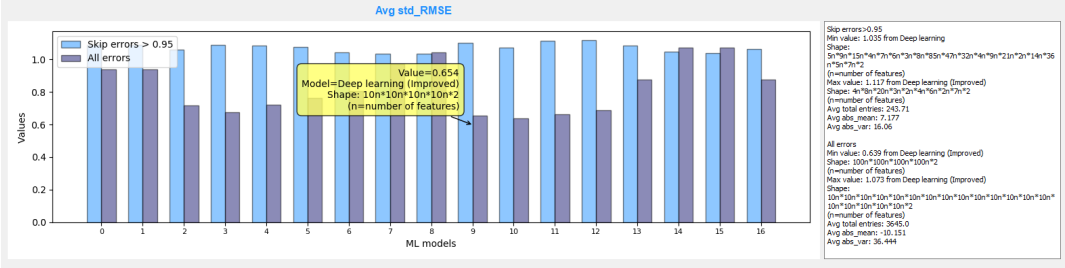


Figura 5.59: Confronto tra i valori di RMSE (Jacobi)

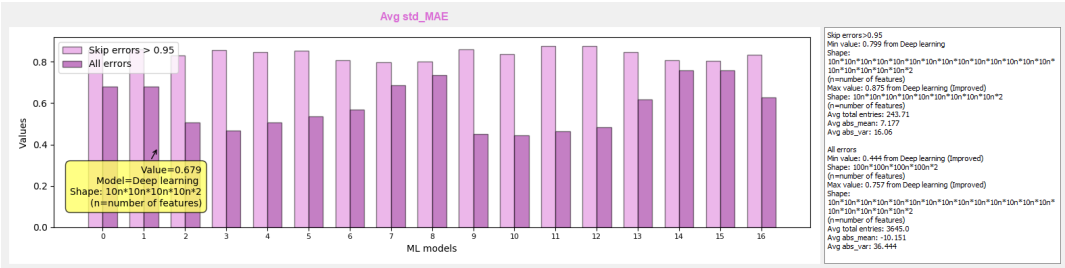


Figura 5.60: Confronto tra i valori di MAE (Jacobi)

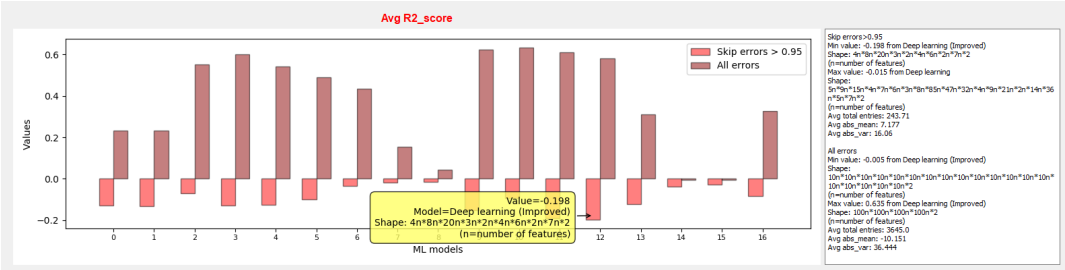


Figura 5.61: Confronto tra i valori di R2_score (Jacobi)

Si conclude perciò affermando che per quanto riguarda benchmark con cardinalità delle variabili esigua o comunque bassa, i modelli studiati (a parte la regressione, sempre inadeguata), approssimano bene la funzione ricercata. Ciò non può essere detto invece per Jacobi e BlackScholes, dove i risultati sono assolutamente insoddisfacenti in tutti i modelli studiati. Infine, in generale si può affermare che i modelli di deep learning risultano essere i migliori nella totalità dei casi, mentre quelli di GCN risultano abbastanza forti in alcuni casi, ma totalmente inadeguati in altri, dove le reti non imparano affatto.

Conclusioni

Con il presente elaborato sono stati implementati diversi modelli di machine learning per individuare, seguendo i principi della Transprecision Computing, il numero di bit minimo sufficiente per il soddisfacimento di requisiti di precisioni specifici.
In particolare, inizialmente sono stati considerati modelli molto semplici, rivelatisi poi inadeguati per la risoluzione del problema: in effetti, gli approcci di regressione lineare sono stati testati fin da subito ma poi sono stati immediatamente esclusi poiché, data la complessità del problema, non sono riusciti ad imparare in maniera precisa la funzione cercata, dando così risultati del tutto insoddisfacenti.
Si è considerato poi l'approccio più preciso del deep learning, testando varie topologie e introducendo incrementalmente diversi numeri di livelli e differenti composizioni di nodi.
Infine, sono stati studiati ed implementati i modelli più moderni delle graph convolutional networks, approccio moderno e ancora piuttosto teorico, però testato nel progetto di tesi al fine di introdurre nei modelli di machine learning i vincoli dati dall'interazione delle variabili dei benchmark finora considerati solo a partire da un grafo.
Inoltre, tali informazioni sono state anche introdotte nelle reti tramite estensione del training set, approccio di più semplice realizzazione ma più oneroso in termini di efficienza computazionale.
In definitiva si può concludere dicendo che per la totalità dei casi gli approcci del deep learning si sono rivelati quelli vincenti poiché hanno dato indici di qualità più soddisfacenti.
Questo però non è stato vero considerando dataset derivanti da benchmark molto più complessi come quelli di BlackScholes e Jacobi, dove nessun approccio è stato in grado di dare buoni risultati.
In tutti gli altri casi, modelli profondi di reti neurali sono stati comunque i migliori, con alcuni casi (Convolution, DWT e Correlation) dove sono stati quasi comparabili ai modelli di GCN (di poco inferiori).
Dunque, la complessità del dominio in esame e della funzione da impara-

re, non ha reso possibile il ritrovamento di un modello adatto per Jacobi e BlackScholes, mentre per gli altri benchmark, i modelli in esame sono stati abbastanza capaci di approssimare la funzione cercata.
Perciò, sebbene gli approcci studiati abbiano dato perlopiù risultati positivi, rimangono ancora punti aperti non trascurabili: in particolare, gli approcci di GCN si sono rivelati ancora troppo acerbi, non garantendo i risultati sperati e lasciando presagire che debbano essere effettuate ancora analisi e ricerche più elaborate al fine di migliorarne le performance.
Inoltre, giunti alla fine di tale elaborato, non ci sono purtroppo ancora modelli in grado di risolvere il problema per quanto riguarda Jacobi e BlackScholes, benchmark molto complessi che hanno perciò bisogno di uno studio ancora più approfondito al fine di trovare un modello predittivo che riesca ad approssimare in una maniera ancora più precisa la funzione cercata, con il vincolo però di non pesare eccessivamente sulle risorse computazionali, problema per nulla banale.

In conclusione, i modelli studiati, seppur non tutti ottimi, rappresentano un valido e robusto punto di partenza per l'implementazione futura di un sistema le cui performance, anche in casi più complessi come quello di Jacobi e BlackScholes, siano comunque ottime.

Bibliografia

[1] Q. Xu, N. Kim, T. Mytkowicz, *Approximate Computing: A survey, IEEE Design and Test*, 2016

[2] Malossi, A.C.I., Schaffner, M., Molnos, A., Gammaitoni, L., Tagliavini, G., Emerson, A., Tomás, A., Nikolopoulos, D.S., Flamand, E., Wehn, N., *The transprecision computing paradigm: Concept, design, and applications*, in "Design, Automation & Test in Europe Conference & Exhibition (DATE)", 2018, pp. 1105–1110

[3] Tavaglini, G., Mach, S., Rossi, D., Marongiu, A., Benini, L., *A Transprecision Floating-Point Platform for Ultra-Low Power Computing*, arXiv:1711.10374, 2017

[4] Kahan, M, *IEEE Standard 754 for Binary Floating Point*, Arithmetic, 1996

[5] N.-M. Ho, E. Manogaran, W.-F. Wong, and A. Anoosheh, *Efficient floating point precision tuning for approximate computing* in "22nd Asia and South Pacific Design Automation Conference (ASP-DAC)", IEEE, 2017, pp. 63–68

[6] S. Graillat, F. Jezequel, R. Picot, F. Fevotte, and B. Lathuiliere, *Autotuning for floating-point precision with Discrete Stochastic Arithmetic*, 2016, [Online], Available https://hal.archives-ouvertes.fr/hal-01331917

[7] Mittal, S., *A Survey Of Techniques for Aproximate Computing* in "ACM Comput. Surv., Vol. a, No. b", Article 1, 2015

[8] Ammar Ben Khadra, M., *An Introduction To Aproximate Computing*, arXiv:1711.06115, 2017

[9] Borghesi, A., Lombardi, M., Milano, M., Tavaglini, G., Benini, L., *Combining Learning and Optimization for Transprecision Computing* (N.P)

[10] Lombardi, M., Milano, M., *Boosting combinatorial problem modeling with machine learning*, in "Proceedings IJCAI", pp. 5472-5478, 2018

[11] Lombardi, M., Milano, M., Bartolini, A., *Empirical decision model learning. Articial Intelligence*, 2017

[12] Viarchi, I., *Active Empirical Model Learning for Transprecision Computing*, 2019 (N.P)

[13] Kipf N. T., Welling M, *Semi-Supervised Classification with Graph Convolutional Networks*, arXiv:1609.02907v4, 2017

[14] Dreiseitl S., Ohno-Machadob L., *Logistic regression and artificial neural network classification models: a methodology review* in "Journal of Biomedical Informatics" Volume 35, Issues 5–6, pp 352-359, 2002

[15] Kaviani P., Dhotre S., *Short Survey on Naive Bayes Algorithm* in "International Journal of Advance Research in Computer Science and Management 04", 2017

[16] Srivastava D. K., Bhambhu L., *Data classification using support vector machine* in "Journal of Theoretical and Applied Information Technology", Vol 12. No. 1, 2010

[17] Breiman L., *Machine Learning*, 2001, https://doi.org/10.1023/A:1010933404324

[18] Soni D., *Supervised vs. Unsupervised Learning*, www.towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d, 2018

[19] Jain A.K., *Data Clustering: 50 Years Beyond K-means*, in "Daelemans W., Goethals B., Morik K. (eds) Machine Learning and Knowledge Discovery in Databases", 2008

[20] Bhande A., *What is underfitting and overfitting in machine learning and how to deal with it*, www.medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76, 2018

[21] Wold S., *Principal component analysis* in "Chemometrics and Intelligent Laboratory Systems", pp 37-52, 1987

[22] García-Gutiérrez J., Martínez-Álvarez F., Troncoso A., Riquelme J. C., *A Comparative Study of Machine Learning Regression Methods on LiDAR Data: A Case Study*, 2013

[23] Lathuilière S., Mesejo P., Alameda-Pineda X., Horaud R., *A Comprehensive Analysis of Deep Regression*, arXiv:1803.08450, 2018

[24] Opitz D., Maclin R., *Popular Ensemble Methods: An Empirical Study*, 1999

[25] Karlik B., Olgac A V., *Performance Analysis of Various Activation Functions in Generalized MLP Architectures of Neural Networks* in "International Journal of Artificial Intelligence And Expert Systems (IJAE)", Volume (1): Issue (4)

[26] Cybenko, G. (1989), *Approximations by superpositions of sigmoidal functions* in "Mathematics of Control, Signals, and Systems", 2 (4), 303-314

[27] Hornik K., *Approximation Capabilities of Multilayer Feedforward Networks*, in "Neural Networks", 4(2), 251–257. doi:10.1016/0893-6080(91)90009-T, 1991

[28] Bruna J., Zaremba W., Szlam A., LeCun Y., *Spectral Networks and Locally Connected Networks on Graphs*, arXiv:1312.6203, 2013

[29] Duvenaud D. K., Maclaurin D., Iparraguirre J., Bombarell R., Hirzel T., Aspuru-Guzik A., Adams R. P., *Convolutional Networks on Graphs for Learning Molecular Fingerprints* in "Advances in Neural Information Processing Systems 28", pp 2224-2232, 2015

[30] Li Y., Tarlow D., Brockschmidt M., Zemel R., *Gated Graph Sequence Neural Networks*, arXiv:1511.05493, 2015

[31] Defferrard M., Bresson X., Vanderghynst P., *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering* in" Advances in Neural Information Processing Systems 29", 2016

[32] Grattarola D., *Spektral*, https://danielegrattarola.github.io/spektral/, 2019

[33] *CORA dataset*, https://relational.fit.cvut.cz/dataset/CORA