Predizione di Docking score con Reti Neurali Ricorrenti

Adlai Santopadre[1]*, Sara Montagna[2]

Sommario
Il docking molecolare rappresenta uno strumento fondamentale nella progettazione di nuovi farmaci, permettendo la stima computazionale dell'affinità tra molecole ligandi e bersagli proteici. In questo studio, ci si concentra sull'approccio ligand-only, in cui la previsione del docking score avviene esclusivamente a partire dalla struttura del ligando, senza informazioni sulla proteina.
È stato sviluppato e testato un modello neurale ricorrente, denominato DockingRNN, basato su LSTM a due strati e allenato su un ampio dataset di molecole rappresentate in SELFIES, un linguaggio robusto e semanticamente vincolato per la codifica molecolare. La strategia implementativa ha incluso tecniche di data augmentation basate su SMILES randomizzati, tokenizzazione dinamica, e ottimizzazione tramite Adam con regolarizzazione (dropout, weight decay) ed early stopping.

Keywords
Deep Learning — Docking score — SELFIES — Model Ligand-only — RNN — LSTM

[1] *Laurea Magistrale in Informatica Applicata, Università degli Studi di Urbino Carlo Bo, Urbino, Italia*
[2] *Docente Applicazioni di Intelligenza Artificiale, Università degli Studi di Urbino Carlo Bo, Urbino, Italia*
***Corresponding author**: a.santopadre@campus.uniurb.it

Introduzione

Il *Docking* molecolare è una tecnica computazionale ampiamente utilizzata nella progettazione di farmaci per stimare la compatibilità tra una molecola (ligando) e una proteina bersaglio. In particolare, il *Docking Score* rappresenta una stima dell'affinità di legame, fornendo un'indicazione quantitativa della probabilità che una molecola si leghi efficacemente a un sito attivo della proteina. Questa metrica è fondamentale nelle prime fasi del *drug discovery*, dove l'obiettivo è identificare candidati promettenti all'interno di grandi librerie molecolari.

Come punto di partenza per l'esplorazione delle tecniche di apprendimento automatico applicate alla chimica computazionale, si è fatto riferimento all'ecosistema DeepChem [1], una libreria open source che integra strumenti per la manipolazione di molecole, l'analisi strutturale e la costruzione di modelli predittivi, costituendo un riferimento consolidato nel dominio del *deep learning for chemistry*.

In questo dominio di ricerca sono state utilizzate le reti neurali ricorrenti (RNN)[2], adatte alla modellazione di sequenze molecolari, specialmente quando rappresentate tramite linguaggi come SMILES(Simplified Molecular-input Line-entry System[3]) o più recentemente SELFIES (SELF-referencIng Embedded Strings), i quali trasformano le strutture chimiche in sequenze simboliche manipolabili da modelli sviluppati in ambito NLP [4].

L'obiettivo perseguito è stato sviluppare un modello *RNN ligand-only* per la predizione del docking score, evitando esplicitamente l'utilizzo della struttura proteica. Questo approccio, sebbene più semplice rispetto al classico *complex-based docking*, si dimostra utile per screening rapidi e per la generazione controllata di molecole attive. Inoltre, si prefigura la possibilità di riutilizzare l'encoder RNN come componente di un *Autoencoder* (AE)[5], al fine di esplorare lo spazio latente delle molecole in maniera generativa e continua.

1. Metodologia

Vediamo quali tecniche e quali scelte sono state adottate per pervenire ad un regressore neurale in grado di predire il docking score a partire dalla sola struttura molecolare del ligando, ignorando la struttura proteica utilizzata. Si parte da un dataset - un file in formato.csv - costituito da coppie ligando, docking score e si sviluppa attraverso diverse fasi: preprocessamento, tokenizzazione, progettazione della rete neurale e addestramento supervisionato.

Listing 1. Analisi del file .csv convertito in dataframe Pandas

```
Data columns (total 2 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   SMILES          226650 non-null  object
 1   Docking_scores  226650 non-null float64
```

1.1 Dataset e rappresentazione molecolare

Come mostrato nel listato_1 il dataset è costituito da una collezione di 226650 ligandi in formato SMILES, ciascuno associato a un valore numerico continuo rappresentante il docking score, una misura dell'affinità al legame espressa in kcal/mol.

Il dataset permette di costruire un modello *ligand-only* per esplorare quanto della predizione possa essere catturato esclusivamente dalla struttura molecolare. Il dataset originario è stato pre-trattato inizialmente per ottenere un dataframe Pandas con due colonne: stringhe ancora in formato SMILES per le molecole e valori docking-scores espressi in floating numbers.

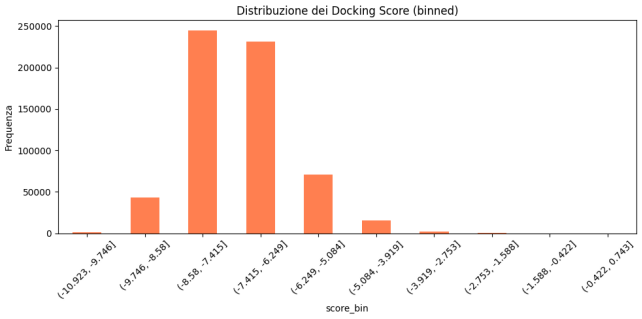

Figura 1. Distribuzione dei valori del docking-score in kcal/mol sul dataset

E' sorta la necessità di poter validare chimicamente la rappresentazione in uso.
La libreria RDKit [6], appositamente dedita alla *cheminformatics* fornisce lo strumento Chem.MolFromSmiles, che nel caso di una stringa SMILES non valida restituisce null. Rispetto al percorso intrapreso dai colleghi [7], orientati dal tipo di rete neurale(CNN e GCNN) alla introduzione delle fingerprints messe a disposizione dalla libreria RDKit, come nuovo punto di partenza per rappresentare le molecole si è optato per il linguaggio **SELFIES** (SELF-referencing Embedded Strings), introdotto da Krenn et al. [8], che costituisce una codifica robusta e garantita rappresentare le strutture molecolari (grafi in origine) tramite stringhe.

Il fondamento di SELFIES risiede in un insieme di regole di derivazione che tengono conto delle proprietà chimiche degli atomi. Dal Supplemento Informativo della citazione [8]:"La rappresentazione SELFIES è una grammatica tipo-2 di Chomsky , context-free, con funzioni autoreferenziali per la generazione valida in rami di grafo". Ad esempio, la stringa SELFIES:

`[F][=C][=C][#N]`

viene interpretata secondo uno stato interno che gestisce la saturazione dei legami, ed è derivata in SMILES come `FC=C=N`, ovvero la molecola 2-fluoroetene-imina. Durante la derivazione, ogni simbolo SELFIES produce un frammento molecolare coerente con i vincoli di valenza chimica (es. il fluoro ha sempre un solo legame, il carbonio può averne fino a 4, ecc.). Per una esemplificazione sulla grammatica di SELFIES si veda la **Appendice**

1.2 Data Augmentation

1.2.1 Comportamento rispetto alle mutazioni

Un punto di forza in SMILES è rappresentato dalla robustezza alle mutazioni casuali.Queste con stringhe SMILES portano molto frequentemente a molecole invalide, mentre in SELFIES la probabilità di validità è pari al 100% anche dopo più mutazioni successive. Questo aspetto risulta cruciale per l'impiego di modelli generativi (come VAE o GAN), in quanto permette di esplorare spazi latenti o combinatori senza incorrere in codifiche illegali.

Per un esempio interattivo che confronti la robustezza di SMILES e SELFIES rispetto a mutazioni casuali, si veda il notebook disponibile su GitHub:
Esempio_Mutazione_smiles.ipynb.
Di seguito l'output del listato di due successive distinte mutazioni (sostituzione di un simbolo dell' alfabeto) a partire dalla stessa molecola FC=C=N rappresentata con SMILE e con SELFIES.

Listing 2. Output del notebook di esempio

```
Original SMILES: FC=C=N
Original SELFIES: [F][C][=C][=N]
Mutated SMILES: FC2C=N -> valido? False
Mutated SELFIES:[F][=N][=C][=N] -> valido?
    True
[05:55:46] SMILES Parse Error: unclosed ring
    for input: 'FC2C=N'
```

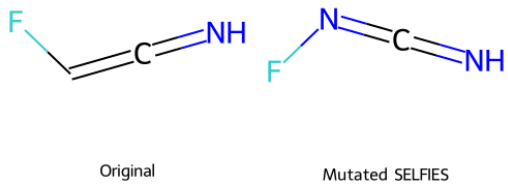

Figura 2. L'alfabeto SELFIES garantisce sempre la validità chimica anche dopo mutazioni casuali.

Al fine di migliorare la generalizzazione del modello e ridurre l'overfitting, è stata implementata una strategia di **Data Augmentation** mediante generazione di SMILES casuali equivalenti per ciascun ligando, poi convertiti in SELFIES,che mantengono la stessa etichetta (il docking score). Una molecola può avere molti SMILES diversi validi, perché SMILES è un linguaggio lineare, ma la molecola è una struttura reticolare tridimensionale e cambiando l'ordine con cui si esplorano gli atomi nel grafo molecolare, si ottengono SMILES diversi. Possiamo ritenere che nel dataset di partenza la rappresentazione valida in uso sia prevalentemente quella cosiddetta `canonica` che si ottiene come risultato deterministico della conversione da RDKit Chem.MolToSmiles(mol). Quello che ci interessa è che, come mostrato nel seguente esempio di codice, da una molecola di partenza si possono ottenere varianti SMILES casuali con il parametro doRandom=True nella conversione. Nel listato è mostrato il risultato della generazione di tre varianti di SELFIES a partire da varianti SMILES. Il docking score associato rimane valido, perché si tratta della

stessa molecola (stessa struttura 3D usata per calcolare lo score) Questa tecnica sfrutta la non unicità della rappresentazione SMILES per ogni molecola, aumentando così la varietà delle sequenze apprese a parità di informazione strutturale.Nella fase di pre-processamento dei dati il campione è stato portato ad una dimensione tripla.

Listing 3. Codice ed output esempio della strategia implementata

```
# Molecola di partenza (es. fluoxetina)
original_smiles = 'CNCCC(O)COc1ccc(C(F)(F)F)
    cc1'
mol = Chem.MolFromSmiles(original_smiles)

# Lista per salvare SELFIES equivalenti
selfies_variants = []

# Genera 3 rappresentazioni SMILES random (
    stessa molecola)
for i in range(3):
    # SMILES random valido (stessa struttura
        )
    smiles_random = Chem.MolToSmiles(mol,
        doRandom=True)
    # Converto in SELFIES
    selfie_str = selfies.encoder(
        smiles_random)

    selfies_variants.append({
        'smiles': smiles_random,
        'selfies_str': selfie_str,

    })

# Stampa le varianti generate
for i, variant in enumerate(selfies_variants
    ):
  print(f"\n### Variante {i+1}")
  print("SMILES:   ",variant['smiles'])
  print("SELFIES:  ",variant['selfies_str'])

### Variante 1
SMILES:          c1cc(OCC(O)CCNC)ccc1C(F)(F)F
SELFIES:         [C][=C][C][Branch1][O][O][C
    ][C][Branch1][C][O][C][C][N][C][=C][C][=C
    ][Ring1][=C][C][Branch1][C][F][Branch1][C
    ][F][F]

### Variante 2
SMILES:          C(NC)CC(COc1ccc(cc1)C(F)(F)F
    )O
SELFIES:         [C][Branch1][Ring1][N][C][C
    ][C][Branch2][Ring1][Branch1][C][O][C][=C
    ][C][=C][Branch1][Branch1][C][=C][Ring1
    ][=Branch1][C][Branch1][C][F][Branch1][C
    ][F][F][O]

### Variante 3
SMILES:          CNCCC(COc1ccc(C(F)(F)F)cc1)O
SELFIES:         [C][N][C][C][C][Branch2][
    Ring1][Branch1][C][O][C][=C][C][=C][
    Branch1][=Branch2][C][Branch1][C][F][
    Branch1][C][F][F][C][=C][Ring1][#Branch2
    ][O]
```

Dall'elenco sono stati rimossi prima dell'addestramento della versione evoluta del modello i duplicati (stringhe identiche) su base SMILES restando con un campione di 607222 ligandi in quanto in fase di training il modello sottoposto più volte allo stesso esempio tende a impararlo " a memoria" , è meno capace in fase di generalizzazione. Inoltre in sede di validazione la presenza di campioni uguali gonfia artificialmente i risultati .

1.3 Tokenizzazione e vocabolario

Nel nostro lavoro, SELFIES è stato dunque utilizzato per codificare tutte le molecole in ingresso al modello DockingRNN. La traduzione SMILES → SELFIES è avvenuta tramite la libreria open-source disponibile su GitHub [**?**]. Successivamente sono state tokenizzate le stringhe dei SELFIES in simboli atomici o strutturali (es. `[C]`, `[Branch1]`, `[Ring2]`, ecc.),estraendo tutte le unità racchiuse tra parentesi quadre semanticamente significative.

Con i token univoci è stato composto l'alfabeto SELFIES. Mediante la sottoclasse Counter() di dict() - contenuta nel modulo *collections* di Python- si è costruito un vocabolario numerico assegnando un indice intero progressivo a ciascun token distinto, inserendo subito al valore 0 il token ¡PAD¿ utile al padding.

Un metodo implementato successivamente è stato il *padding* dinamico: per ogni batch, le sequenze numeriche sono state allungate fino alla lunghezza massima del batch stesso legata alla sequenza più lunga, evitando padding eccessivo e migliorando l'efficienza computazionale. I simboli di padding a fine stringa sono stati poi ignorati al momento dell'elaborazione. Di questo si è occupato il codice della funzione *collate*.

1.4 Dataset Pytorch e splitting casuale dei dati

I dati a disposizione sono stati suddivisi casualmente in tre insiemi secondo un criterio di suddivisione classico in:

1. Set di training pari all'80% del set complessivo
2. Set di validazione pari al 10%
3. Set di test pari al 10%

Tali set sono stati manipolati con le funzioni DockingDataset() e istanziati con DataLoader() di Pytorch la libreria Python utilizzata per il modello.

2. Architettura del modello DockingRNN

Le RNN sono reti neurali progettate per lavorare con dati di tipo (temporale) sequenziale. A differenza delle reti *feedforward*, mantengono una memoria dello stato precedente, il che le rende ideali per elaborare input sequenziali come testo, audio o sequenze molecolari (es. SELFIES). Le RNN semplici soffrono di problemi di *vanishing gradient*: di conseguenza col passare dei time step, diventa difficile apprendere dipendenze a lungo termine[9].

Il modello neurale implementato, denominato *DockingRNN*, è basato su una rete **LSTM** (Long Short-Term Memory) [10],che affronta il vanishing gradient ed è particolarmente adatta a sequenze molecolari.

Le LSTM sono un tipo speciale di RNN che introduce una struttura interna chiamata ”cell state” capace di trasportare le informazioni lungo la sequenza. Le LSTM usano tre ”porte” per controllare il flusso delle informazioni:

- Forget gate: decide cosa dimenticare
- Input gate: decide cosa aggiungere alla memoria
- Output gate: decide cosa mandare in output

Secondo le seguenti equazioni di aggiornamento:

- $f_t = \sigma(W_f\, x_t + U_f\, h_{(t-1)} + b_f\,)$
  $i_t = \sigma(W_i\, x_t + U_i\, h_{(t-1)} + b_i\,)$
  $\widetilde{c_t} = \tanh\left(W_c x_t + U_c h_{t-1} + b_c\right)$
  $c_t = f_t * c_{t-1} + i_t * \widetilde{c_t}$
  $o_t = \sigma\left(W_o x_t + U_o h_{(t-1)} + b_o\right)$
  $h_t = o_t * \tanh\left(c_t\right)$

le cui relazioni sono evidenziate nel grafico in figura sottostante

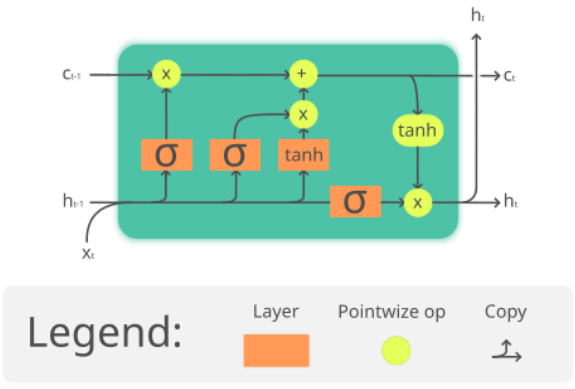

Figura 3. Una cella LSTM - fonte Guillaume Chevalier, CC BY-SA 4.0 ¡https://creativecommons.org/licenses/by-sa/4.0¿, via Wikimedia CommonsEnter

L’architettura del modello neurale costruito comprende i blocchi che andremo ad elencare.

Listing 4. schema

```
SELFIES sequence
--> [Embedding Layer]
--> [1st LSTM Layer (hidden_dim)]
--> [2nd LSTM Layer (hidden_dim)]
--> [Last hidden state h_n (from 2nd LSTM)]
--> [Linear -> docking score]
```

Un livello di embedding che trasforma ogni token numerico in un vettore denso.Il modulo torch.nn.Embedding è uno strato di lookup usato per convertire indici discreti (interi) in vettori densi appresi dal modello. Fa uso della sintassi *embedding = nn.Embedding(num_embeddings, embedding_dim)* dove num_embeddings è la dimensione del vocabolario (numero di token unici) e embedding_dim la dimensione dei vettori che rappresentano ogni token 4

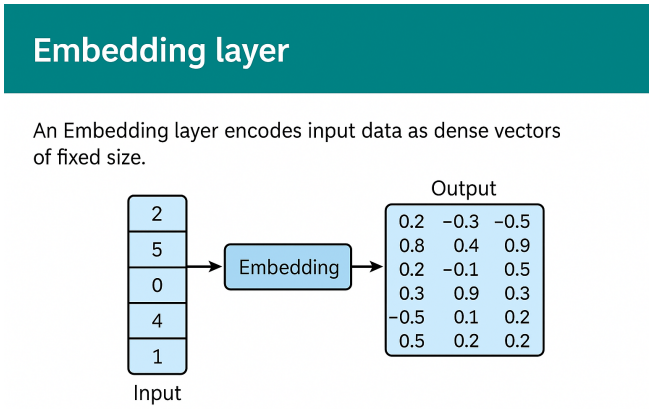

Figura 4. Embedding Layer

Gli strati LSTM sono utilizzati per l’elaborazione sequenziale e l’apprendimento di pattern - nel nostro caso sottostrutture chimiche ricorrenti e interazioni non strettamente locali dovute ad anelli, ramificazioni. Il modello è stato irrobustito ponendo num_layers=2 ovvero stratificando due LSTM in modo che l’output del primo strato divenisse l‘input del secondo; Un livello fully connected per restituire un valore continuo. In input riceve la estrazione dello *hidden state finale* e fornisce il valore per la predizione;

Il metodo costruttore del modello _init_ riceve come parametri:

- vocab_size: numero di token nel vocabolario SELFIES
- embedding_dim: dimensione dello spazio vettoriale in cui i token verranno mappati (= 128)
- hidden_dim: dimensione del vettore nascosto nell’LSTM (=256)
- dropout: percentuale di dropout per regolarizzazione

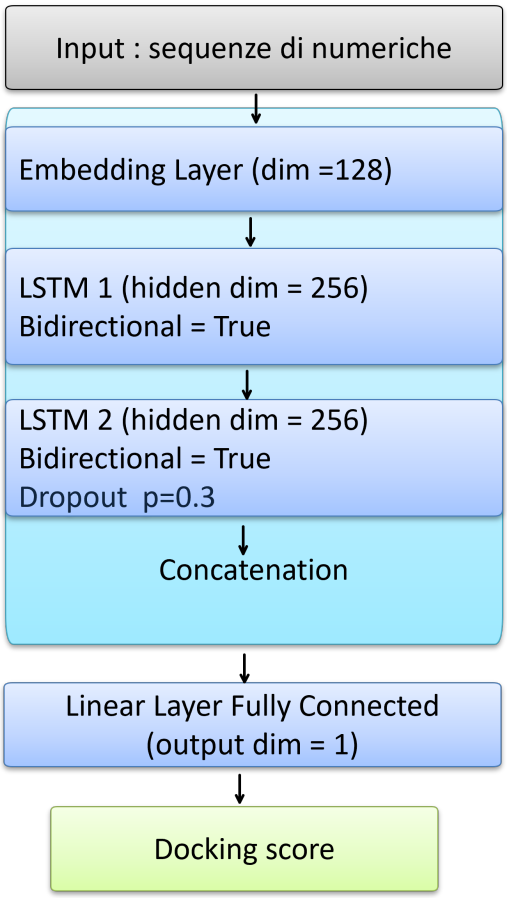

Figura 5. Diagramma rete neurale

Le ulteriori modifiche via via sperimentate (documentate nella sezione Risultati) sono state:

- l'implementazione di bidirectional=True un parametro che fa sì per ogni LSTM si procedesse in avanti e indietro nel percorrere le sequenze - trattandosi di un dominio il cui risultato era carpire quanto più contesto della struttura molecolare;
- adozione del dropout per prevenire l'overfitting combinato con weight_decay (parametro di regolarizzazione L2 utilizzato con l'ottimizzatore scelto *Adam*) per penalizzare i pesi troppo grandi . itemize

2.1 Funzione obiettivo e metrica

Il modello DockingRNN è stato addestrato con la **Mean Squared Error (MSE)** come funzione di loss. Tale scelta è motivata dalla sensibilità del modello alle deviazioni nel docking-score reali, trattandosi, nell'ultimo stadio di un modello che opera come regressore.

A scopo valutativo, sono state inoltre monitorate:

1. lA RMSE (Radice quadrata dell'Errore quadratico medio)
2. la MAE (Errore Assoluto medio) che deve tendere a valori bassi nel senso di una maggiore accuratezza del modello
3. la metrica R^2 che indica la proporzione della varianza nei dati dipendenti spiegata dal modello. Un R^2 vicino a 1 suggerisce che il modello spiega bene la variabilità dei dati; un valore vicino a 0 indica il contrario

2.2 Addestramento e Validazione

Come già detto si è operato con dataset suddiviso in set di **train**, **validation** e **test** secondo uno schema 80-10-10 riutilizzando sempre gli stesso set. L' algoritmo di ottimizzazione **Adam**, ha operato con *learning rate* lr=1e-3 costante, come suggerito per esso. Per contrastare l'overfitting e' stato testato uno script di tuning operante su valori combinati di dropout in [0.2, 0.3, 0.4] e valori di weight_decay in [0.2, 0.3, 0.4].

Addestramento e validazione sono stati condotti ricorrendo alla tecnica dell' *Early stopping* inserendo un monitoraggio del valore di loss della validazione (valore che deve essere decrescente), un valore di *patience* ovvero un contatore di epoche rispetto all' ultimo miglioramento della valore di loss, e il ricorso al salvataggio dello stato corrispondente al best model , recuperabile in caso di superamento del numero di epoche senza miglioramento , come risultato dell'addestramento. Nel codice di training loop seguito dalla evaluation, è stato implementato l' aggiornamento della funzione obiettivo e delle metriche ad ogni epoca (loss, MAE, RMSE, R²)

3. Risultati raggiunti

Vengono di seguito presentati e commentati i risultati delle sessioni di addestramento in relazione alla evoluzione del modello. Siamo partiti da un Dataset iniziale aumentato a **679500** campioni suddiviso in: Train: 543960 — Val: 67995 — Test: 67995

Modello "DockingRNN" di partenza: 1 layer LSTM addestrato con early stopping (patience = 3); ottimizzatore Adam con lr=1e-3
Miglior modello salvato: Epoch 22 — Train Loss: 0.5666 — Val Loss: 0.6422.

Modello "DockingRNN" con *2 layers LSTM e dropout = 0.3* addestrato con early stopping (patience = 3); ottimizzatore Adam con lr=1e-3 *weight_decay=1e-5*
Miglior modello salvato: Epoch 21 — Train Loss: 0.5193 — Val Loss: 0.5918 Per questo addestramento è presente plot del docking score predetto vs quello reale) Fig 6

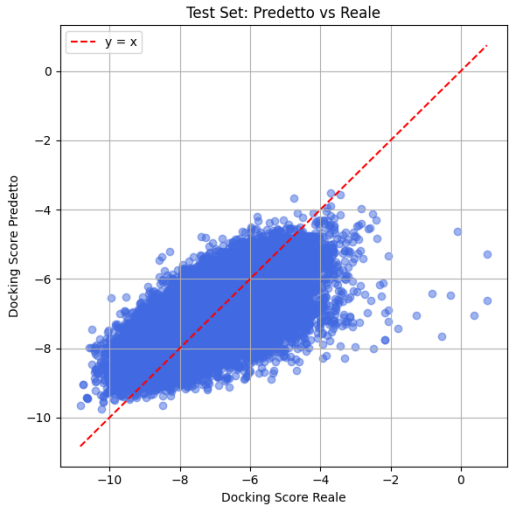

Figura 6. score Predetto Vs score Reale

Il Dataset iniziale è stato ripulito di 43244 campioni duplicati identici (nuovo dataset di 202774 campioni aumentato a **607222** suddivisi in : Train: 486657 — Val: 60832 — Test: 60833

Lo stesso modello ”DockingRNN” è stato riaddestrato ottenendo risultati migliori; qui si riportano anche le metriche adottate di cui si parla in dettaglio più avanti Epoch 15 — Train Loss: 0.5152 — Val Loss: 0.5709 — Train MAE: 0.5527 — Val MAE: 0.5800 — R^2: Train 0.4811 — Val 0.4136

Il Modello ”DockingRNN” (con 2 layer LSTM e dropout =0.3 addestrato con early stopping (patience = 3); ottimizzatore Adam con lr=1e-3 weight_decay=1e-5) è stato implementato come biLSTM attivando l‘opzione *bidirectional = True* con i seguenti risultati
Epoch 18 — Train Loss: 0.4889 — Val Loss: 0.5651 — Train MAE: 0.5394 — Val MAE: 0.5740 — R^2: Train 0.5076 — Val 0.4195 e relativo plot fig 6

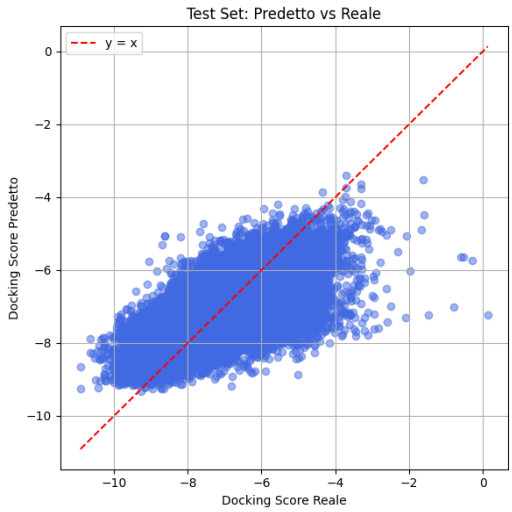

Figura 7. Predetto Vs score Reale -biLSTM

Invero come anticipato nel precedente paragrafo, il modello finale ”DockingRNN” con 2 layer LSTM e bidirectional = True è stato sottoposto a test con combinazione di valori di dropout e weight_decay) Nei risultati della sottostante Tabella 1 si osserva come il modello beneficia di una regolarizzazione soft: poco dropout (0.2–0.3) e weight decay basso (1e-5).

Dropout	Weight Decay	Final Val Loss	Epochs
0.2000	0.00001	0.5890	17
0.2000	0.0001	0.6961	11
0.2000	0.001	0.7879	12
0.3000	0.00001	0.5993	16
0.3000	0.0001	0.6592	20
0.3000	0.001	0.9751	16
0.4000	0.00001	0.5890	17
0.4000	0.0001	0.6736	14
0.4000	0.001	0.8843	2

Tabella 1. loop di addestramento con Dropout e Weight_decay differenti

In seguito è stato sperimentato un passaggio di post processing. Calcolato l‘errore assoluto tra predizione e dato reale sono stati scartati i campioni peggiori eliminando i residui oltre la soglia del percentile 95 (eliminati 24333 esempi del train set di 486657 campioni potenziali *outliers*).Il modello è stato riaddestrato con dropout = 0.3 ottenendo: Epoch 11 — Train Loss: 0.5138 — Val Loss: 0.5832 — Train MAE: 0.5662 — Val MAE: 0.5802 — R^2: Train 0.4556 — Val 0.4010 Risultati non migliorativi rispetto alla attesa.

3.1 Interpretazione dei Grafici

Vengono riportati grafici di interesse per l’interpretazione dei risultati che fanno riferimento all’addestramento dell’ultima evoluzione del modello, prima dell’esclusione di campioni ulteriori in seguito al filtraggio dei residui.

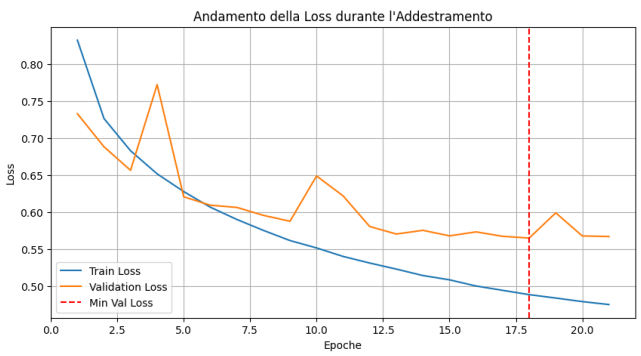

Figura 8. Andamento delle loss durante le epoche

Confronto Train vs. Validation Loss da fig 8 : L'analisi di queste curve mostra visivamente come se la perdita di addestramento continua a diminuire mentre quella di validazione si stabilizza o aumenta, si manifesta indicare *overfitting*, Perciò ha senso arrestare il training con la tecnica dell'early stopping. La linea verticale rossa indica l'epoca in cui la perdita di validazione è stata minima, suggerendo il miglior punto per interrompere l'addestramento.

Quanto alle metriche che abbiamo adottato , il valor medio dello scarto, la radice quadrata dell' errore quadratico medio e il cosidetto R^2, la successiva figura mostra l'evoluzione in fase di addestramento e di validazione durante le epoche , dei valori calcolati.

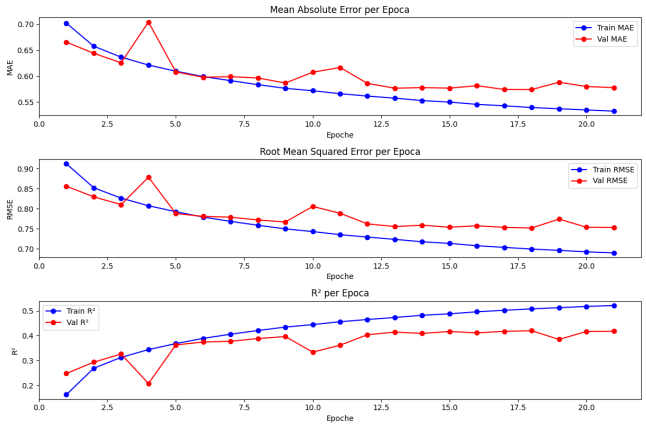

Figura 9. MAE , RMSE , e R^2 a confronto

Per i primi due grafici (MAE e RMSE)osserviamo come la diminuzione di queste metriche (con il beneficio della attesa di tre epoche consecutive sfavorevoli prima di arrestarsi, indica che il livello di miglioramento nell'accuratezza delle predizioni. Quando le curve di validazione iniziano ad aumentare mentre quelle di training continuano a diminuire e nella fattispecie oltre la 18 esima epoca, è sintomo di overfitting. Il modello raggiunge dei risultati significativi rispetto alla complessità computazionale non elevata rispetto ad esempio a modelli di rete convolutive o grafico-convolutive, e in assenza di una iniezione di feature engineering in cui alla rappresentazione della struttura del ligando si possono associare in input caratteristiche chimico-fisiche per il dominio di interesse Il valore in ascesa per la terza metrica indica quanto bene il modello stia spiegando una proporzione sempre maggiore della varianza nei dati. Valori stabili o in diminuzione durante le epoche possono indicare underfitting o overfitting, a seconda del contesto.

3.2 Risultati della applicazione del modello sul dati dataset di test

Calcolando il docking-score sui dati riservati al test conclusivo e calcolando di nuovo rispetto ai valori noti reali i valori per le metriche in uso sono stati ottenuti questi dati:

• Il modello, in media, sbaglia di circa 0.62 kcal/mol • A seconda della scala dei tuoi docking scores, questo può essere accettabile o migliorabile • Esempio: se i docking vanno da -10 a -4, 0.6 è un errore medio non trascurabile

Test RMSE: 0.7541 • Siccome RMSE è superiore al valore del MAE il modello è più sensibile agli outlier commettendo errori più marcati su alcune molecole.

Test MAE: 0.5742 Il modello, in media, sbaglia di circa 0.57 kcal/mol. Per valori di docking che oscillano vanno da -10 a -4, 0.6 è un errore medio non trascurabile ma una buona base di studio

Test R^2: 0.4236 • Il modello spiega 42.7 della varianza nei dati

4. Approfondimenti e sviluppi possibili

L'ottimizzazione dell'attuale modello può passare attraverso sperimentazioni con l'aggiunta di feature del dominio chimico disponibili tramite RDKit (LogP, MW, TPSA...)
In maniera paradossale rispetto ai motivi che hanno fatto preferire SELFIES a SMILES - specialmente la coerenza semantica - si possono prendere in considerazione SMILES random non validati per aumentare la variabilita' del dataset. D'altra parte, verità di dibattito ci porta a citare il paper recente "Invalid SMILES are beneficial rather than detrimental to chemical language models" [11] che supporta l' uso di rappresentazioni semanticamente imperfette in input.

Il modello si presta a prelevare l'hidden state prima della regressione finale e ad analizzarlo con UMAP oppure t-SNE (vedi esemplificazione del notebook con input il dataset di test disponibile su GitHub).
Questo tipo di analisi corrisponde all'analisi dello *text spazio latente* utilizzata per evidenziare features implicite.
Il modello di rete RNN può essere utilizzato come *encoder* nella progettazione di un Variational AutoEncoder[12][13] e per esplorazione molecolare come compito successivo alla regressione[14] .

Infine l'interesse potrebbe essere rivolto ad esplorare il potenziale dello spazio latente a fini generativi tra-

mite l'implementazione di meccanismi di attention e l' adozione di Transformer[15],

Riferimenti bibliografici

[1] Zhenqin Wu, Bharath Ramsundar, Evan N. Feinberg, Joseph Gomes, Cody Geniesse, Aneesh S. Pappu, Ken Leswing, and Vijay S. Pande. Deepchem: An open-source toolkit for deep learning in drug discovery, quantum chemistry, materials science and biology. `https://deepchem.io`, 2024. Accessed: 2025-05-13.

[2] Robin Winter, Floriane Montanari, Andreas Steffen, Hans Briem, Frank Noé, and Djork-Arné Clevert. Learning continuous and data-driven molecular descriptors by translating equivalent chemical representations. *Chemical Science*, 10(6):1692–1701, 2019.

[3] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.*, 28:31–36, 1988.

[4] Zhenqin Xu, Shuang Wang, Fei Zhu, and Jianfeng Huang. Seq2seq fingerprint: An unsupervised deep molecular embedding for drug discovery. *arXiv preprint arXiv:1711.07957*, 2017.

[5] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamin Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, 2018.

[6] Landrum. Rdkit: Open-source cheminformatics. *Journal of chemical information and modeling*, 2006.

[7] Attaranto Roselli. Utilizzo di cnn e gnn per la predizione dei docking score a partire da strutture molecolari. *https://github.com/GioRoss/Utilizzo-di-CNN-e-GNN-per-la-Predizione-dei-Docking-Score-a-Partire-da-Strutture-Molecolari/blob/main/Relazione.pdf*, 2024.

[8] Mario Krenn, Florian Häse, Animesh Nigam, Pascal Friederich, and Alán Aspuru-Guzik. Selfies: a robust representation of semantically constrained graphs. *Machine Learning: Science and Technology*, 1(4):045024, 2020.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[11] Michael A. Skinnider. Invalid smiles are beneficial rather than detrimental to chemical language models. *Nature Machine Intelligence*, 2024.

[12] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2014.

[13] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. *arXiv preprint arXiv:1703.01925*, 2017.

[14] Akshat Nigam, Roberto Pollice, Mario Krenn, et al. Beyond generative models: Superfast traversal, optimization, novelty, exploration and discovery (stoned) algorithm for molecules using selfies. *Journal of Chemical Information and Modeling*, 61(6):2777–2790, 2021.

[15] Shion Honda, Shao Shi, and Hiroyuki R Ueda. Smiles transformer: Pre-trained molecular fingerprint for low data drug discovery. *arXiv preprint arXiv:1911.04738*, 2019.