POLITECNICO DI TORINO

Collegio di Ingegneria Chimica e dei Materiali

Corso di Laurea Magistrale
in Ingegneria Chimica e dei Processi Sostenibili

Tesi di Laurea Magistrale

Accoppiamento di Machine Learning e Dinamica Molecolare: applicazioni di reti neurali a simulazioni in solvente implicito di proteine

Relatori
prof. Roberto Pisano
prof. Gianluca Boccardo
dott. Andrea Arsiccio

Candidato
Federica Ruà

Anno accademico 2021/2022

Indice

1 Introduzione

L'intelligenza artificiale (*Artificial Intelligence*, AI) viene definita come la scienza e l'ingegneria del costruire macchine intelligenti, nata ufficialmente alla conferenza di Dartmouth del 1956, ed è da decenni considerata una delle tecnologie più preminenti [1]. L' intelligenza artificiale è quindi utilizzata per simulare l'intelligenza umana allo scopo di aiutare nel *problem solving* e nel compiere decisioni al posto degli esseri umani; le sue applicazioni spaziano tra differenti campi come l'economia, l'ingegneria, il diritto e la medicina [2].

L'applicazione di AI ha due principali rami: virtuale e fisica. La componente fisica fa principalmente riferimento alle sue applicazioni nel campo della robotica mentre la componente virtuale è rappresentata dal *Machine Learning* (ML) che consiste in algoritmi matematici che migliorano il loro apprendimento tramite l'esperienza [2].

1.1 Machine Learning

Il Machine Learning è definito da Arthur Samuel (1959) come: *Machine learning is the field of study that gives computers the ability to learn without being explicitly programmed*, ossia quel campo di studi che dona ai computer la capacità di imparare senza essere esplicitamente programmati. L'obiettivo del Machine Learning è quindi quello di trovare algoritmi e modelli statistici che un computer può utilizzare a questo fine [3]. Una definizione più recente è stata proposta da Tom Mitchell (1998): *A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E*, ovvero un computer impara dall'esperienza quando nel portare a compimento dei compiti migliora la sua performance nel compiere suddetti compiti grazie all'esperienza [4].

Il *Machine learning* è stato visto come una delle soluzioni a un problema che da decenni affliggeva lo studio di sistemi complessi. Come Paul Dirac affermava nel 1929,[5], lo studio di sistemi complessi in ambito chimico e fisico è limitato non dalla conoscenza della teoria matematica alla base, oramai in gran parte nota, ma dalla difficile applicazione di questa teoria, che porta a equazioni troppo complicate da risolvere. Risulta necessario sviluppare metodi approssimati ma pratici che possano risolvere ed identificare le proprietà principali di un sistema complesso senza esagerati costi computazionali.

Il *Machine Learning* ha permesso quindi di sviluppare metodi approssimati per sistemi complessi senza ricadere nella problematica di equazioni troppo complicate da potere risolvere [6].

I differenti algoritmi di ML si dividono essenzialmente in tre grandi categorie:

- *Supervised Learning*, è il compito di imparare una relazione che collega un *input*, dato in entrata del modello, con l' *output*, dato in uscita. Gli algoritmi di *supervised learning* richiedono quindi un intervento esterno, nella forma di un set di dati *input-output* forniti all'algoritmo.

- *Unsupervised Learning*, gli algoritmi sono lasciati a sé stessi nel dedurre e presentare correlazioni e strutture interessanti presenti nei dati analizzati, in questo modo gli algoritmi imparano nuove proprietà dai dati, e quando nuovi dati sono introdotti le proprietà precedentemente individuate vengono utilizzate per classificare le nuove serie di dati.

- *Reinforcement Learning*, in cui il programma di apprendimento prende delle decisioni in base all'ambiente con il fine di massimizzare delle possibili 'ricompense' (*rewards*) [3] .

1.2 Reti neurali

Tra gli algoritmi più celebri nell'ambito del ML vi sono le Reti Neurali, (*Artificial Neural Networks*, ANNs) che sono alla base del *Deep Learning*. Sono versatili, potenti e scalabili e ciò le rende perfette per affrontare grandi e complicati compiti nel mondo del ML. Le ANNs sono state utilizzate per classificare immagini (es. Google Images), sistemi di riconoscimento vocale (es. Siri di Apple) o raccomandazioni sui video che meglio rispecchiano le preferenze dell'utente (es. YouTube), o allo scopo di sconfiggere i giocatori umani più forti al mondo nel gioco del go, come nel famosissimo caso di AlphaGo creato da Google DeepMind [4][1].

Utilizzando come riferimento il sistema nervoso umano, e nello specifico il neurone, è facile intuire il concetto di base su cui si è lavorato per introdurre il concetto di ANN e il suo modellarsi. I neuroni sono cellule presenti nella corteccia vertebrale animale e sono composti da una corpo cellulare contente il nucleo, da prolungamenti ramificati che si dipartono dal corpo chiamati dendriti, e infine da un prolungamento decisamente più lungo chiamato assone che nella sua estremità si ramifica in telodendri che presentano le sinapsi, strutture che si collegano ai dendriti di altri neuroni (Figura 1). Le sinapsi fanno sì che i neuroni possano scambiarsi fra di loro segnali tramite impulsi elettrici, quando un neurone riceve un numero sufficiente di segnali per unità di tempo è esso stesso in grado di trasmettere un segnale ai neuroni a cui è collegato. La struttura semplice, ma ramificata dei neuroni è in grado di eseguire delle operazioni complesse, per via del fatto che ogni singolo neurone

è collegato a migliaia di altri neuroni generando uno dei sistemi più complessi presenti in natura [4].

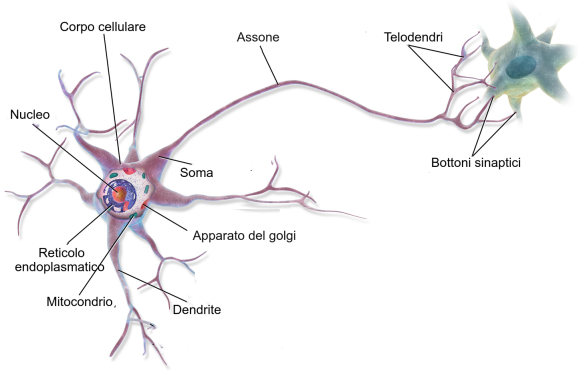


Figura 1: Schema di un neurone. Figura tratta da [4] con modifiche.

Similarmente ai neuroni nel sistema nervoso, la *Threshold Logic Unit* (TLC) è l'unità di base nelle reti neurali. Più TLU unite a formare un unico *layer* danno origine a quello che è definito percettrone, quando ogni singolo neurone è connesso a tutti i neuroni di un *layer* precedente si ha quello che è definito *dense layer* cioè un *layer* totalmente connesso. Nel caso si uniscano vari percettroni si ottiene una struttura definita *Multi Layer Perceptron* (MLP) (Figura 2), che presenta un *input layer*, uno o più layer composti da percettroni, chiamati *hidden layer*, e il *layer* finale che è definito *output layer* [4].

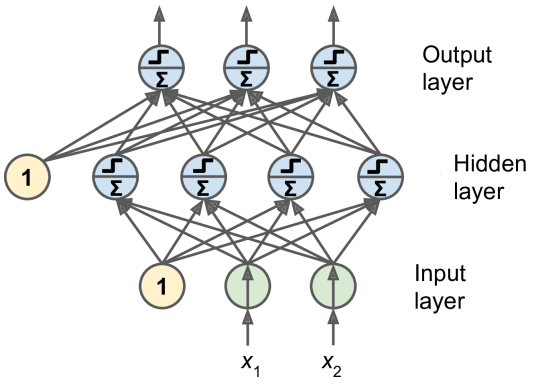


Figura 2: *Multi Layer Perceptron*, i simboli in ogni neurone indicano: la somma pesata dei pesi ($\sum$), il tipo di funzione di attivazione, in questo esempio la funzione a gradino, eq.(3). Figura tratta da [4] con modifiche.

1.2.1 Elementi teorici delle ANN

L' unità fondamentale alla base delle ANNs, come precedemente accennato, è la TLC che calcola la somma pesata (attraverso dei pesi $\mathbf{w}^T = (w_1, w_2, ..., w_n)$) dei valori numerici ($\mathbf{x}= (x_1, x_2, ..., x_n)$) ricevuti in entrata, *inputs*, e poi applica una funzione di attivazione (f_{att}). Il risulato di questo calcolo è l'output della TLC (h_w).

$$z = w_1x_1 + w_2x_2 + .. + w_nx_n = \mathbf{w^T x} \tag{1}$$

$$h_w(\mathbf{x}) = f_{att}(z) = f_{att}(\mathbf{w^T x}) \tag{2}$$

Le principali funzioni di attivazione sono :

- funzione a gradino di Heavyside:

$$\begin{cases} 0\, se\, z < 0 \\ 1\, se\, z \geq 0 \end{cases} \tag{3}$$

- funzione segno, alternativa alla funzione a gradino:

$$\begin{cases} -1\, se\, z < 0 \\ 0\, se\, z = 0 \\ 1\, se\, z > 0 \end{cases} \tag{4}$$

- funzione logistica:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{5}$$

- funzione tangente iperbolica:

$$\tanh z = 2\sigma(2z) - 1 \tag{6}$$

- funzione ReLU (*Rectified Linear Unit*):

$$ReLU(z) = max(0, z) \tag{7}$$

L'insieme di più TLC in un unico *layer*, come precedentemente illustrato, dà origine al percettrone. Ad ogni *layer* viene aggiunto un nuovo elemento,

che è il neurone *bias*, con *output* uguale a 1 e che non riceve *input* (Figura 3) [4].

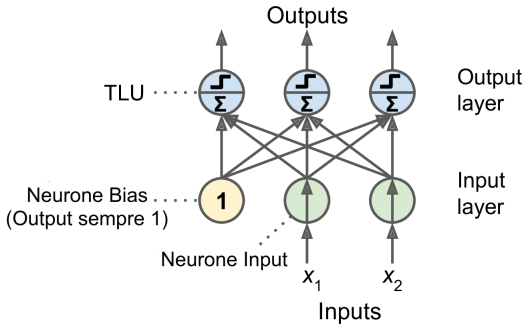


Figura 3: *Struttura di un Percettrone.* Figura tratta da [4] con modifiche.

I calcoli effettuati dal percettrone si distinguono da quelli effettuati dalle singole TLC per la presenza del *bias* b_j.

$$z_1 = w_1 x_1 + w_2 x_2 + .. + w_j, x_j + .. + b_1 \tag{8}$$

$$z_j = w_1 x_1 + w_2 x_2 + .. + w_j, x_j + .. + b_j \tag{9}$$

$$z_N = w_1 x_1 + w_2 x_2 + .. + w_j, x_j + .. + b_N \tag{10}$$

$$h_{\mathbf{W},\mathbf{b}}(\mathbf{X}) = f_{att}(\mathbf{XW} + b) \tag{11}$$

Le reti neurali, quando vengono applicate in un contesto di *Supervised Learning*, nel quale gli *input* vengono suddivisi in due tipi di *dataset*, *training* e *test dataset*, hanno un *workflow* che prevede una fase iniziale definita *training*. In questa specifica fase il rispettivo dataset viene processato dal modello in modo da ottenere delle predizioni, mentre il *test dataset* è riservato a una fase di validazione del modello [3].

Il percettrone nella fase di *training* processa un campione di dati alla volta e al termine di ogni campione deve tenere conto dell'errore commesso dalla rete, ovvero di quanto i valori predetti dalla rete si discostano dai valori di riferimento. Questo si rende possibile considerando che due neuroni sono tanto più connessi, quanto più grandi sono i pesi che li collegano: variando i valori dei pesi si va ad incidere sul livello di connessione tra due neuroni. Quindi più un neurone contribuisce all'errore commesso della rete, tanto più

i suoi pesi verranno alleggeriti, ragionamento opposto per neuroni che invece diminuiscono l'errore.

L'equazione 12 illustra l'algoritmo che regola i valori dei pesi durante la fase di *training* di uno specifico campione di dati: $w_{i,j}$ è il peso che collega l'i-esimo neurone di input e lo j-esimo neurone di output, z_i è l'output del neurone i-esimo mentre t_i è il valore target. Il parametro λ che appare nell' equazione è il *learning rate* che stabilisce quanto velocemente la rete adatta i propri parametri ai nuovi dati. Il *learning rate* fa parte di un insieme di parametri che vengono definiti iperparametri che determinano la struttura di una rete neurale e la dinamica numerica del suo addestramento [4].

$$w_{i,j}^{k+1} = w_{i,j}^{k} + \lambda(t_i - z_i)x_j \tag{12}$$

Quando nella rete neurale non si ha un singolo percettrone ma multipli percettroni impilati si ha, come già accennato, un *Multi Layer Perceptron*, che appartiene alla categoria delle *Deep Neural Networks* (DNN). Il training di una DNN sfrutta un algoritmo di retropropagazione. Similarmente al singolo percettrone, l'algoritmo calcola per ogni singolo campione di dati un valore di *output*, e ne valuta l'errore, nonché il contributo di ogni connessione neurale sull'errore, e di conseguenza aggiorna i pesi in modo da minimizzare l'errore. Vi sono differenti algoritmi di retropropagazione, uno dei più comuni è il *gradient descent*, in cui si variano iterativamente i parametri al fine di minimizzare una funzione errore. Il concetto che sta alla base di questo algoritmo è il calcolo del gradiente locale della funzione errore rispetto ai pesi e ai *bias*, e si procede nella direzione del gradiente massimo finché il gradiente è nullo, ovvero si è minimizzato l'errore. All'inizio di ogni *training* i valori assegnati ai pesi e ai *bias* sono casuali, ma in seguito vengono modificati iterativamente al fine di minimizzare l'errore [4].

1.3 Dinamica molecolare

Le tecniche di *Machine learning*, *Artificial Neural Networks* e *Deep Neural Networks* finora discusse appartengono alla categoria dei modelli definiti *data-driven*, ovvero si basano su di un set di dati e per un ottimale funzionamento di questi modelli il *dataset* di partenza deve avere dimensioni considerevoli e contenere una notevole quantità di dati. Ottenere enormi volumi di dati non è sempre agevole in settori come l'ingegneria chimica, ma l'introduzione di simulazioni facilita l'ottenimento di *dataset* importanti. In questo contesto un ruolo importante è svolto dalle simulazioni di dinamica molecolare (*Molecular Dynamics*, MD) , che servono da ponte di collegamento tra la scala microscopica di atomi e molecole e il mondo del macroscopico

del laboratorio. Partendo da ipotesi sull'interazione tra le molecole si ottengono predizioni sulle proprietà del *bulk*; l'accuratezza delle predizioni può essere manipolata infittendo il livello di dettaglio della descrizione impiegata, tenendo però conto delle capacità computazionali dell'elaboratore. Le simulazioni servono da ponte anche tra il mondo teorico e quello sperimentale, per esempio conducendo simulazioni con modelli teorici per poi confrontare i risultati ottenuti con quelli sperimentali. La dinamica molecolare consente di studiare le proprietà di un insieme di molecole, in termini di struttura e delle interazioni microscopiche tra di esse. Simulazioni di dinamica molecolare permettono quindi di studiare l'evoluzione di un sistema fisico e chimico a livello atomistico e molecolare [7].

La dinamica molecolare consiste essenzialmente nella risoluzione numerica, *step-by-step*, dell'equazione classica del moto, dove la forza conservativa $\mathbf{f}_i$, descrivibile come il gradiente del potenziale $\mathcal{U}$, agisce su un atomo di massa m_i e posizione $\mathbf{r_i}$ generando su di esso un'accelerazione $\ddot{\mathbf{r}}_\mathbf{i}$:

$$m_i \ddot{\mathbf{r}}_\mathbf{i} = \mathbf{f}_i \qquad \mathbf{f}_i = -\frac{\partial}{\partial \mathbf{r_i}} \mathcal{U} \tag{13}$$

Alla base della risoluzione dell'equazione del moto vi è quindi la conoscenza delle forze agenti sul sistema. Le forze sono derivate dall'energia potenziale $\mathcal{U}(\mathbf{r}^N)$, dove $\mathbf{r}^N = (\mathbf{r}_1, \mathbf{r}_2, ...\mathbf{r}_N)$ rappresenta il vettore delle $3N$ coordinate atomiche [7].

1.3.1 Elementi teorici di Dinamica Molecolare

La conoscenza delle forze e quindi dell'energia potenziale del sistema, analizzato nella simulazione di dinamica molecolare, risulta il passaggio chiave. Nel considerare le interazioni molecolari e le rispettive energie potenziali si può procedere ad una distinzione fondamentale tra *non-bonded interactions* e *bonded interactions*, ossia tra interazioni agenti tra atomi non legati in modo covalente, ed interazioni che si instaurano invece tra atomi consecutivi nella struttura molecolare considerata.

- *Non-bonded Interactions*
  L'energia potenziale $\mathcal{U}_{non-bonded}$ è tradizionalmente suddivisa nei termini *1-body*, *2-body*, *3-body* ..., ovvero si suddivide l'energia potenziale in base al numero di atomi coinvolti nella singola interazione

  $$\mathcal{U}_{non-bonded}(\mathbf{r}^N) = \sum_i u(\mathbf{r}_i) + \sum_i \sum_{j>i} v(\mathbf{r}_i \mathbf{r}_j) + ... \tag{14}$$

  Il termine $u(\mathbf{r})$ rappresenta campi potenziali applicati dall'esterno ed in genere le interazioni tra più di due atomi sono trascurate a favore

delle interazioni di coppia $v(\mathbf{r}_i\mathbf{r}_j)$ [7].
I potenziali più comunamente utilizzati sono:

- Potenziale di Lennard-Jones v^{LJ}, funzione della distanza r tra i due atomi coinvolti:

$$v^{LJ}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{15}$$

dove σ è il diametro dell' atomo o della sfera rigida che approssima la molecola, ed ε la profondità della buca di potenziale (Figura 4).

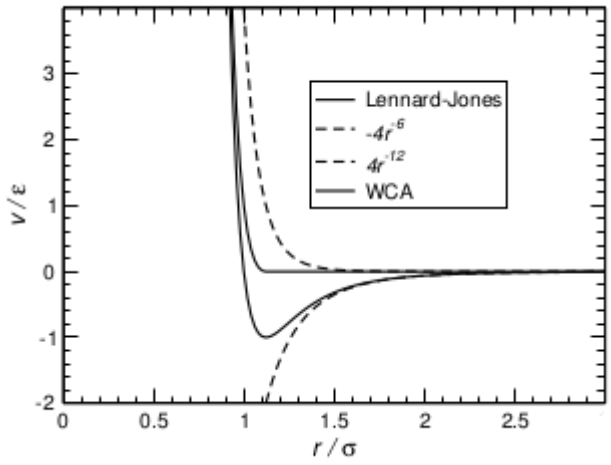


Figura 4: Potenziale di Lennard-Jones. Figura tratta da [7] con modifiche.

- Potenziale di Coulomb:
si tratta del termine potenziale da considerare in caso di presenza di cariche elettriche

$$v^{Coulomb}(r) = \frac{Q_1 Q_2}{4\pi\varepsilon_0 r} \tag{16}$$

dove Q_1, Q_2 sono le cariche ed ε_0 è la costante dielettrica nel vuoto [7] .

- *Bonded Interactions*
L'energia potenziale $\mathcal{U}_{bonded}$ tiene in considerazione le interazioni intramolecolari collegate ai legami chimici presenti tra gli atomi. Si

considera quindi un'energia potenziale strettamente intramolecolare.

$$\begin{aligned}\mathcal{U}_{bonded} = & \frac{1}{2}\sum_{legami} k^r_{ij}(r_{ij}-r_{eq})^2 \\ & + \frac{1}{2}\sum_{angoli\,di\,piega} k^\theta_{ijk}(\theta_{ijk}-\theta_{eq})^2 \\ & + \frac{1}{2}\sum_{angoli\,di\,torsione}\sum_m k^{\phi,m}_{ijkl}(1+\cos(m\phi_{ijkl}-\gamma_m))\end{aligned} \tag{17}$$

Per i legami tra coppie adiacenti di atomi è stato considerato un potenziale armonico definito dalla costante k^r_{ij}, con una specifica distanza di equilibrio r_{eq} tra i due atomi i e j coinvolti. Gli angoli di piega θ sono angoli tra terne di atomi, mentre gli angoli di torsione ϕ fanno riferimento alle interazioni tra quaterne di atomi (Figura 5) [7]. Per gli angoli di piega si utilizza in genere un potenziale armonico, definito dalla costante k^θ_{ijk} e dall'angolo di equilibrio θ_{eq}, mentre il potenziale torsionale è in genere rappresentato come una sequenza di coseni con molteplicità m, fase γ_m ed ampiezza definita dalla costante $k^{\phi,m}_{ijkl}$.

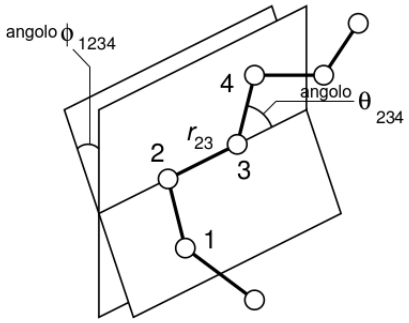


Figura 5: Geometria di una catena di molecole con evidenziate la distanza tra due atomi r_{23} e gli angoli di piega θ_{123} e di torsione ϕ_{1234}. Figura tratta da [7] con modifiche.

Per fare delle considerazioni sull'equazioni del moto, in un sistema di atomi, si studia l'equazione riscritta in termini di quantità di moto $\mathbf{p}^N = (\mathbf{p}_1, \mathbf{p}_2, ...\mathbf{p}_N)$ e delle coordinate $\mathbf{r}^N = (\mathbf{r}_1, \mathbf{r}_2, ...\mathbf{r}_N)$.

$$\dot{\mathbf{r}}_i = \mathbf{p}_i/m_i \qquad \dot{\mathbf{p}}_i = \mathbf{f}_i \tag{18}$$

Questo è un sistema di due equazioni differenziali accoppiate, caratterizzate dal fatto di essere *stiff* ('rigide'), ovvero i metodi di soluzione sono numericamente instabili a meno che il passo d'integrazione ∂t impiegato sia estremamente piccolo. L'algoritmo che meglio si adatta alla risoluzione di queste equazioni è l'algoritmo di Verlet [7].

1.3.2 L'algoritmo di Verlet

Esistono varie forme, considerate essenzialmente equivalenti, dell'algoritmo di Verlet:

- Integrazione di Verlet:

$$\mathbf{r}(t+\partial t) = 2\mathbf{r}(t) - \mathbf{r}(t-\partial t) + \ddot{\mathbf{r}}_\mathbf{i}\partial t^2 \tag{19}$$

- Integrazione *Leap-frog*:

$$\mathbf{r}(t+\partial t) = \mathbf{r}(t) - \mathbf{v}\left(t+\frac{1}{2}\partial t\right)\partial t \tag{20}$$

$$\mathbf{v}\left(t+\frac{1}{2}\partial t\right) = \mathbf{v}\left(t-\frac{1}{2}\partial t\right) + \ddot{\mathbf{r}}_\mathbf{i}(t)\partial t \tag{21}$$

dove $\mathbf{v}$ è il vettore velocità.

- Velocità di Verlet:

$$\mathbf{r}(t+\partial t) = \mathbf{r}(t) + \mathbf{v}(t)\partial t + \frac{1}{2}\ddot{\mathbf{r}}_\mathbf{i}\partial t^2 \tag{22}$$

$$\mathbf{v}(t+\partial t) = \mathbf{v}(t) + [\ddot{\mathbf{r}}_\mathbf{i}(t) + \ddot{\mathbf{r}}_\mathbf{i}(t+\partial t)]\frac{\partial t}{2} \tag{23}$$

Le caratteristiche fondamentali dell'algoritmo di Verlet sono le seguenti: è esattamente reversibile rispetto al tempo; è simplettico, ovvero garantisce la conservazione dell'energia; è di basso ordine rispetto al tempo permettendo quindi passi di integrazione ampi; richiede un unico calcolo della forza per ciascuna iterazione; è facile da programmare [7].

Tendenzialmente si cerca di non rappresentare i legami intramolecolari nella forma di una funzione di energia potenziale, siccome tali legami hanno elevate frequenze vibrazionali e dovrebbero quindi essere trattati attraverso approssimazioni della meccanica quantistica e non di quella classica. I legami sono quindi vincolati a una lunghezza fissa, tramite dei *constraints*, che nella meccanica classica sono introdotti tramite i formalismi langragiani o hamiltoniani. Gli schemi che implementano i *constraints* sono chiamati SHAKE [8], nel caso si applichi l'algoritmo d'integrazione originale di Verlet, o RATTLE [9], nel caso in cui si applichi l'algoritmo velocità di Verlet [7].

1.3.3 Simulazioni a solvente esplicito o implicito

L'ambiente solvatato intorno a macromolecole, come proteine, assume un ruolo fondamentale e a volte decisivo sia nella struttura che nella dinamica dei sistemi biologici. Ad esempio, la presenza di un 'nucleo idrofobico', struttura presente in varie proteine, influenza fortemente il *folding* di una proteina in una soluzione acquosa. Il solvente inoltre rende le strutture delle proteine abbastanza flessibili per cambi conformazionali e media le interazioni tra macromolecole durante i processi biologici. Per questo motivo, per simulazioni di dinamica molecolare di sistemi biologici, risulta fondamentale ottenere modelli accurati che descrivano gli effetti dovuti alla presenza di molecole di solvente. In simulazioni MD esistono due principali metodi di solvatazione: esplicito e implicito. Il primo incorpora esplicitamente le molecole di solvente nel sistema simulato, mentre il secondo rappresenta gli effetti del solvente integrandoli nel campo di forze che agisce sulla molecola [10]. Uno dei metodi per le simulazioni MD in solvente implicito più comunemente utilizzato è il modello *Generalized-Born* (GB) che approssima le interazioni elettrostatiche tra due atomi, i e j, a lungo raggio, con la formula:

$$E_{ij}^{elec} = E_{ij}^{vac} + E_{ij}^{solv} \tag{24}$$

$$E_{ij}^{vac} = \frac{q_i q_j}{r_{ij}} \qquad (i < j) \tag{25}$$

$$E_{ij}^{solv} = -\frac{1}{2}\left[\frac{1}{\epsilon_{in}} - \frac{\exp(-0.73 k f_{ij}^{GB})}{\epsilon_{out}}\right]\frac{q_i q_j}{f_{ij}^{GB}} \tag{26}$$

$$f_{ij}^{GB} = \sqrt{r_{ij}^2 + B_i B_j e^{-r_{ij}^2/4B_iB_j}} \tag{27}$$

dove E_{ij}^{elec}, E_{ij}^{vac} ed E_{ij}^{solv} sono i rispettivi contributi energetici totale, nel vuoto, e di solvatazione, dovuti all'interazione elettrostatica tra i due atomi i e j a distanza r_{ij}. I termini q_i e q_j rappresentano le cariche dei due atomi, ϵ_{in} e ϵ_{out} le costanti dielettriche del soluto e del solvente. B_i e B_j sono i raggi di Born degli atomi e k è il parametro di *screening* di Debye-Hückel [11].

Il metodo a solvente implicito presenta vari vantaggi: accelera i calcoli per determinare le forze riducendo drasticamente il numero di gradi di libertà, aumenta la dimensione effettiva di un *time step* nelle simulazioni MD e semplifica le simulazioni a pH costante [11]. Nonostante i vantaggi, l'accuratezza dei modelli a solvente implicito tende ad essere inferiore rispetto al modello a solvente esplicito.

1.3.4 Machine Learning in simulazioni di dinamica molecolare

Una delle prime applicazioni del *Machine Learning* in chimica è stata l'estrazione di superfici di energia potenziale (classiche) da calcoli di meccanica quantistica. L'opera seminale di Behler e Parrinello [12], in questa direzione, ha aperto la porta ad oggigiorno a un campo di ricerca in costante crescita. L'applicazione di algoritmi di ML è stata introdotta, a seguito di questi primi approcci in campo chimico, anche nel mondo della dinamica molecolare, per l'analisi e la simulazione di traiettorie. Ad esempio, applicando modelli di ML si sono ottenuti: *force-fields* atomistici [13], modelli molecolari *coarse-grained* [14], stime su superfici di energia libera [15], coordinate di reazione per tecniche di sampling amplificato [16]. Questi differenti aspetti delle simulazioni di MD sono stati sviluppati in modo autonomo, e ad esempio la generazione di *force-fields* tramite ML si è concentrata su molecole di piccola dimensione mentre l'analisi di traiettorie di MD è maggiormente pertinente a simulazioni di molecole grandi e flessibili come le proteine. L'applicazione di strumenti di *Machine Learning* in ambito di simulazioni di dinamica molecolare rimane quindi un campo in cui resta molto da approfondire e migliorare [6].

1.4 Obiettivi del lavoro di tesi

Nel presente lavoro di tesi si è voluto accoppiare i risultati di simulazioni di dinamica molecolare con modelli surrogati di *machine-learning* basati su reti neurali. Tramite simulazioni di dinamica molecolare si è ottenuto un ampio *dataset*, e successivamente a passaggi di *pre-processing* i dati così ottenuti sono stati forniti come *input* e *output* di riferimento alle reti neurali. Si è quindi soddisfatto la necessità di modelli *data-driven*, nello specifico di reti neurali, di voluminose quantità di dati da processare grazie a simulazioni al computer di dinamica molecolare e allo stesso tempo si è cercato di sfruttare le capacità predittive dell'ANN per predire l'evoluzione di proprietà di un sistema di atomi senza dover ricorrere unicamente agli algoritmi di dinamica molecolare.

L'obiettivo del lavoro è stato quello di cercare di utilizzare le reti neurali per predire proprietà quali le coordinate degli atomi dopo un certo numero di passi di integrazione, le forze istantanee e le forze medie, e di fare considerazioni sulla qualità delle predizioni tramite confronti con gli stessi dati ottenuti tramite le simulazioni di MD.

Il lavoro si è concentrato su due casi studio:

I caso studio:
Si sono effettuate simulazioni di MD in solvente implicito della proteina

$(AAQAA)_3$ e si sono utilizzati i dati ottenuti per il *trainig* della rete neurale.

II caso studio:
Si sono effettuate simulazioni di MD in solvente implicito del dipeptide di alanina e si sono utilizzati i dati ottenuti per il *training* della rete neurale.

Il manoscritto verrà strutturato nel seguente modo:

- esposizione dei dettagli riguardanti le simulazioni di MD dei due casi studio ed esposizioni dei calcoli di *pre-processing* sui dati ricavati al fine di ottenere un *dataset* per entrambe le proteine;

- presentazione del modello ANN e delle sue specifiche, nonché delle fasi di *training* e *test* delle reti neurali e delle varie metodologie utilizzate.

- esposizione e analisi dei risultati.

- confronto con algoritmi presenti in letteratura [14].

- conclusioni.

L'utilizzo di modelli di *machine learning* come le reti neurali su *dataset* ottenuti da simulazioni di MD e preprocessati *offline* si presenta come un possibile punto di partenza per un eventuale utilizzo delle stesse reti neurali duranti le simulazioni, ovvero in modalità *online*, con la finalità di sostituire gli algoritmi di MD, in alcune iterazioni, ed i costosi calcoli computazionali legati alla determinazione delle forze agenti sul sistema.

2 Creazione del dataset

La creazione del *dataset*, step fondamentale per modelli *data-driven*, è ottenuta in questa tesi tramite simulazioni di dinamica molecolare; da quest'ultime si possono estrarre dati contenenti informazioni riguardanti le coordinate, le velocità e le forze del sistema di atomi coinvolti nelle simulazioni.

2.1 Simulazioni di dinamica molecolare

Le simulazioni di MD sono state eseguite per mezzo del programma di simulazioni biomolecolari AMBER 20 [17]. Il programma mette a disposizione differenti versioni di *Force-Field*, termine che nell'ambito della meccanica molecolare fa riferimento impropriamente sia alla forma funzionale che al set di parametri utilizzati per esprimere l'energia potenziale di un sistema di particelle. Nel presente lavoro si è deciso di utilizzare il *force-field* AMBER99SB-disp [18]. Si sono simulati due diversi sistemi: la proteina $(AAQAA)_3$ e la dialanina. Entrambe le proteine sono state studiate a 298 K.

2.1.1 Minimizzazione energetica

In una prima fase l'energia potenziale del sistema oggetto di studio è stata minimizzata, prima di passare alla simulazione vera e propria in solvente implicito. Questa prima fase serve a ridurre le forze agenti sul sistema e a garantire il buon esito delle simulazioni successive. Infatti, l'assenza di una appropriata minimizzazione della struttura può portare ad instabilità durante le integrazioni dell'equazione del moto da parte del programma di MD. Abbiamo utilizzato il *tool* sander di AMBER per effettuare 3000 passi di minimizzazione, che permettono di avvicinarsi al punto di minimo locale più prossimo alla struttura di partenza, utilizzando l'algoritmo della discesa del gradiente (*steepest descent*).

2.1.2 Simulazioni in solvente implicito

L'obiettivo del presente lavoro è stato quello di simulare entrambe le succitate proteine ($(AAQAA)_3$ e dialanina) in ambiente acquoso. Esistono due approcci per tener conto della presenza del solvente in MD: 1) simulazioni a solvente esplicito, in cui le molecole del solvente sono esplicitate e le varie interazioni del solvente con la molecola di interesse sono determinate ad ogni singolo step; 2) simulazioni a solventi implicito, in cui gli effetti dovuti alla presenza del solvente sono integrati a priori nelle equazioni del *force-field*.

Si è scelto di svolgere, per entrambi i casi in studio, simulazioni a solvente implicito, per evitare l'eccessivo costo computazionale legato alla presenza

esplicita del solvente. Il metodo di solvente implicito implementato in AMBER è il modello *Generalized Born* (GB)(indicato come IGB in sander), nello specifico si è scelto di utilizzare una versione modificata del modello GB (IGB = 5) [19].

Le simulazioni in solvente implicito sono state svolte per ambedue i casi studio, come già accennato precedentemente, alla temperatura di 298 K. Per il controllo della temperatura si sono utilizzate le dinamiche di Langevin, con una frequenza di collisione di 1 ps^{-1}. Si sono imposti dei *constraints* sui legami che coinvolgono gli atomi di idrogeno, tramite l'applicazione dell'algoritmo SHAKE [8]. Inoltre, non si è imposto un raggio di *cut-off* per le interazioni di Lennard-Jones e di Coulomb.
Si è utilizzato un *time-step* di 2 fs, e i dati di coordinate, velocità e forze istantanee sono stati salvati ogni 50 iterazioni.

Il programma di MD richiede come informazione anche le velocità iniziali dei singoli atomi, che vengono generate in modo casuale in base alla distribuzione di Maxwell-Boltzmann alla temperatura selezionata per la simulazione.

2.1.3 Caso studio proteina $(AAQAA)_3$

Nel primo caso studio la molecola presa in esame è la proteina $(AAQAA)_3$ (Figura 6), proteina formata da 180 atomi e che prende il suo nome dalla sequenza dei residui proteici Alanina-Alanina-Glutammina-Alanina-Alanina ripetuta tre volte nella struttura della proteina (Figura 7).

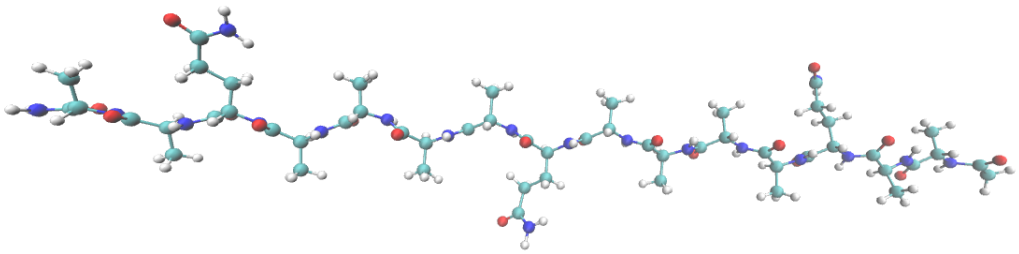


Figura 6: Proteina $(AAQAA)_3$.

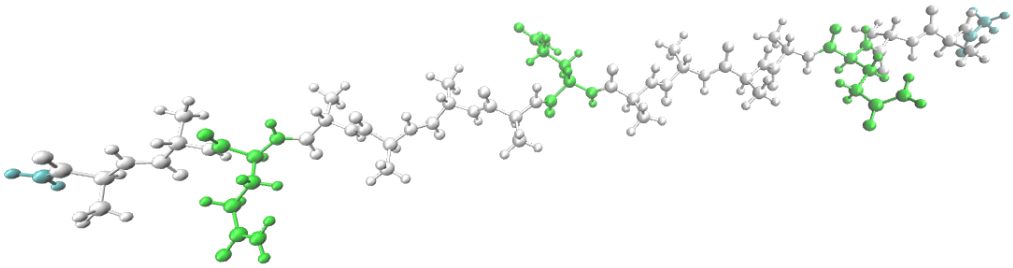


Figura 7: Rappresentazione della proteina $(AAQAA)_3$ con i residui proteici evidenziati: in bianco l'alanina, in verde la glutammina.

Per questo caso studio si è voluto studiare l'evoluzione della proteina per un lasso totale di tempo di 10 ns (ovvero per un totale di 5×10^6 iterazioni). Il *dataset* preliminare generato conteneva perciò 10^5 *time-step*. Durante la simulazione si è utilizzato in combinazione ad AMBER 20 il programma Plumed 2.4.7 [20], al fine di monitorare i valori di raggio di girazione e di contenuto di alfa-eliche. Ciò consente di fare delle considerazioni sull'evoluzione della struttura della proteina nel lasso di tempo considerato.

Il raggio di girazione è stato definito come:

$$Raggio\,di\,girazione \qquad : R_g = \sqrt{\frac{\sum_{i=1}^{N} m_i (r_i - r_C)^2}{\sum_{i=1}^{N} m_i}} \tag{28}$$

il termine r_C fa riferimento alla posizione del centro di massa, mentre r_i e m_i sono le posizioni e masse dei singoli atomi.
Il contenuto di alfa-eliche della proteina è determinato dal numero di sezioni di residui aventi configurazione ad alfa-elica, attraverso la seguente equazione [21]:

$$\alpha - Elica\,(componente) \qquad : \alpha = \sum_{\mu} g\left[r_{dist}(\{R_i\}_{i\in\Omega_\mu}, \{R^0\})\right] \tag{29}$$

dove $\{R_i\}_{i\in\Omega_\mu}$ sono le coordinate atomiche di un set Ω_μ di 6 residui della

proteina, mentre $g(r_{dist})$ è la seguente funzione,

$$g(r_{dist}) = \frac{1 - \left(\frac{r_{dist}}{r_0}\right)^8}{1 - \left(\frac{r_{dist}}{r_0}\right)^{12}} \tag{30}$$

dove r_{dist} è la distanza (RMSD, *root-mean-square deviation*) rispetto a una configurazione alfa-elica di riferimento $\{R^0\}$, mentre r_0 una distanza *cut-off* di 0.08 nm.

La proteina è stata introdotta all'inizio della simulazione nella sua configurazione non ripiegata (*unfolded*). Nell'intervallo di tempo considerato è possibile osservare la proteina ripiegarsi, assumendo una struttura secondaria ad alfa-elica [22].

Infatti, il raggio di girazione (Figura 8) della molecola decresce velocemente all'inizio della simulazione, per poi oscillare intorno a un valore di equilibrio per il resto del tempo considerato. Il grafico del contenuto di alfa-eliche evidenzia a sua volta una rapida formazione della struttura secondaria (Figura 9).

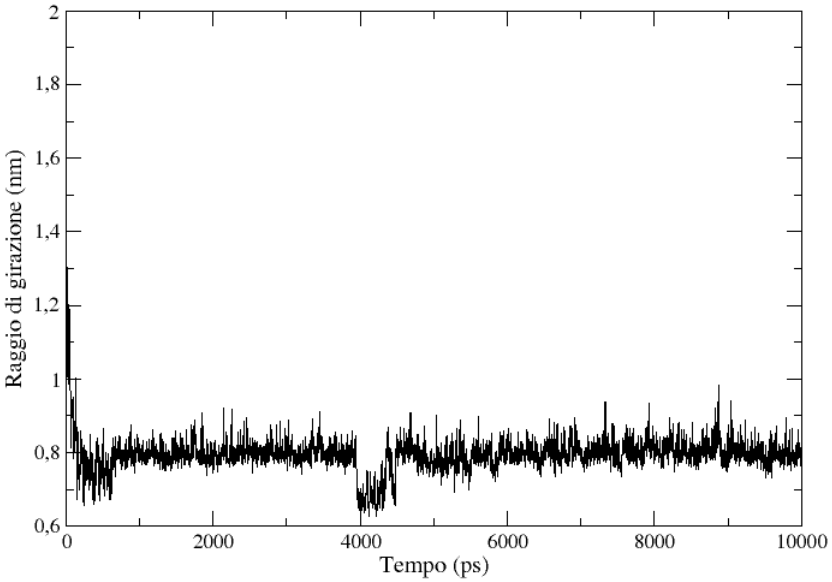


Figura 8: Raggio di girazione della proteina $(AAQAA)_3$.

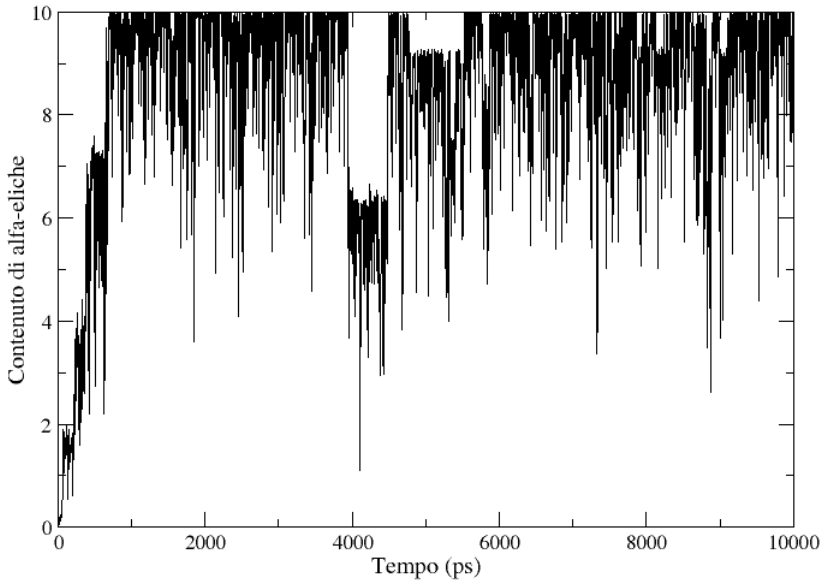


Figura 9: Contenuto di alfa-eliche nella struttura secondaria della proteina $(AAQAA)_3$.

2.1.4 Caso studio proteina alanina dipeptide

Nel secondo caso studio la proteina presa in considerazione è l'alanina dipeptide, anche detta dialanina (Figura 10), molecola composta da due residui alaninici uniti da un legame peptidico per un totale di 23 atomi.

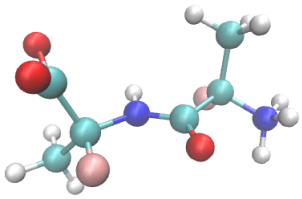


Figura 10: Alanina dipeptide o dialanina.

Nel presente caso studio si è deciso di simulare la proteina per un totale di 400 ns, salvando i dati ogni 50 *time-step*, ottenendo così un *dataset* di 4×10^6 punti.

Durante la simulazione si è operata una valutazione tramite il grafico di Ramachandran [23]. Quest'ultimo mostra la distribuzione delle varie strutture secondarie a seconda dell'ampiezza degli angoli diedri ψ e ϕ posti rispettivamente sull'asse y e x, come mostrato in Figura 11 [24]. Un angolo diedro, in chimica, è l'angolo compreso tra due piani passanti tra due set di tre atomi aventi in comune due atomi. Nel *backbone* di una proteina è possibile definire tre angoli diedri:

- ω, l'angolo formato dagli atomi C^α - C - N - C^α

- ϕ, l'angolo formato dagli atomi C - N - C^α - C

- ψ, l'angolo formato dagli atomi N - C^α - C - N (originariamente chiamato ϕ' da Ramachandran).

dove C^α è il carbonio alfa che nelle molecole organiche si riferisce al primo atomo di carbonio che si lega a un gruppo funzionale. La planarità del legame peptidico impone che l'angolo diedro ω sia pari a 180° (configurazione *trans*) e in rari casi 0° (configurazione cis) [25].

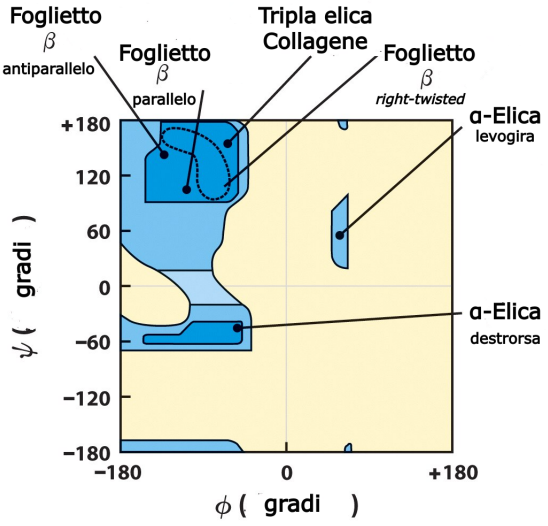


Figura 11: Esempio di un grafico di Ramachandran. Immagine tratta da [26] con modifiche.

Tramite il grafico di Ramachandran applicato al nostro caso studio si è potuto constatare che, nel lasso di tempo monitorato durante la simulazione,

la dialanina assumeva delle specifiche strutture secondarie, tra cui oscillava nel tempo (Figura 12).

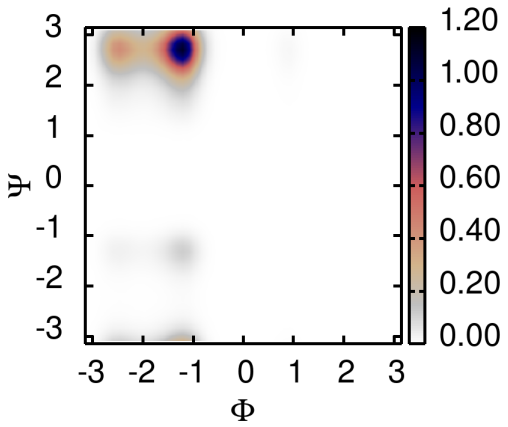


Figura 12: Grafico di Ramachandran dell'alanina dipeptide calcolato durante gli ultimi 300 ns della simulazione. I primi 100 ns sono stati considerati come un'equilibratura, e scartati nella fase di analisi. Gli angoli sono in radianti.

Per motivi legati ad una gestione più agevole del *dataset* della dialanina, se ne è ridotta la dimensione a 49999 *frame*, estratti dopo il primo milione di *time-step* salvati.

2.2 Preprocessing dei dati: invarianza alla traslazione e rotazione

Una questione chiave nell'applicare modelli *data-driven* a sistemi fisici è fino a che punto la conoscenza del sistema oggetto di studio debba essere integrata negli algoritmi. Una scuola di pensiero nella comunità del *machine learning* è che solo il minimo indispensabile di informazioni debba essere implementato negli algoritmi, e forniti un sufficiente numero di dati, il modello dovrebbe essere in grado di distinguere strutture e *patterns* da solo. Ciò si basa sulla convinzione che implementare delle nozioni fisiche nei modelli di *machine learning* potrebbe essere rischioso nel caso in cui la nozione implementata risultasse difettosa, con conseguenti perdite a livello di *performance* del modello.

In molti sistemi fisici, comunque, esistono conoscenze pregresse nella forma di invarianze e simmetrie [27][28] . L'invarianza o simmetria, in fisica, è la proprietà posseduta da alcune grandezze di non essere modificate, sotto l'azione di una trasformazione nello spazio e/o nel tempo, dall'applicazione della trasformazione stessa. Le trasformazioni possono essere ad esempio: traslazioni spaziali, temporali o rotazioni. Le proprietà di simmetria (o di invarianza) sono intimamente legate alle leggi di conservazione. Il teorema di Noether, infatti, afferma che a ogni simmetria continua in una teoria di campo lagrangiana corrisponde una quantità conservata (e viceversa) [29]. L'invarianza a traslazioni spaziali corrisponde per esempio a conservazioni della quantità di moto, mentre l'invarianza a rotazioni corrisponde ad una conservazione del momento angolare.

Nella letteratura, nonostante il crescente interesse nell'applicazione di modelli *data-driven* a sistemi fisici, non esiste un consenso su come le proprietà degli *input* debbano essere comunicate alla rete, e sul modo in cui gli *input* debbano rispettare le proprietà invarianti del sistema. Se tradizionalmente l'invarianza è esplicitamente considerata, i modelli di *machine learning* hanno anche l'opzione di usare *input* non invarianti ed eseguire il *training* del modello affinché esso impari anche le proprietà invarianti [30].

Nel presente lavoro si è voluto garantire l'invarianza alla traslazione e rotazione delle coordinate, velocità e forze istantanee ricavate dalle simulazioni di dinamica molecolare di entrambi i casi studio. L'approccio è stato quello di non lasciare il compito di imparare a riconoscere le proprietà invarianti al modello, ma di processare i dati ricavati dalle simulazioni, gli *input*, al fine di garantire tali proprietà di invarianza anche negli output [30].

2.2.1 Invarianza alla traslazione

Per garantire l'invarianza alla traslazione del *dataset* si è dovuto traslare il centro del sistema di riferimento nel centro di massa della molecola. Il centro di massa o baricentro di un corpo rigido costituito da N masse è il punto geometrico corrispondente al valor medio della distribuzione delle masse, che formano il corpo rigido, nello spazio:

$$x_{CM} = \frac{\sum_{i=1}^{N} m_i x_i}{\sum_{i=1}^{N} m_i} \tag{31}$$

$$y_{CM} = \frac{\sum_{i=1}^{N} m_i y_i}{\sum_{i=1}^{N} m_i} \tag{32}$$

$$z_{CM} = \frac{\sum_{i=1}^{N} m_i z_i}{\sum_{i=1}^{N} m_i} \tag{33}$$

dove (x_{CM}, y_{CM}, z_{CM}) sono le coordinate del centro di massa, mentre (x_i, y_i, z_i) sono le coordinate della massa i-esima, m_i.

Le coordinate sono quindi state ricalcolate rispetto al centro di massa:

$$x'_i = x_i - x_{CM} \tag{34}$$

$$y'_i = y_i - y_{CM} \tag{35}$$

$$z'_i = z_i - z_{CM} \tag{36}$$

dove (x'_i, y'_i, z'_i) sono le coordinate della massa i-esima traslate rispetto al centro di massa. Grazie a questo *pre-processing* dei dati delle coordinate si è potuta garantire l'invarianza alla traslazione negli *output* della rete neurale.

2.2.2 Invarianza alla rotazione

Per garantire l'invarianza alla rotazione del *dataset* è stato necessario effettuare un cambiamento di base, dal sistema di riferimento precedentemente traslato nel centro di massa, al sistema di riferimento avente come assi gli assi principali d'inerzia della molecola.

Per un corpo nello spazio è sempre possibile trovare tre assi mutualmente ortogonali per i quali i prodotti d'inerzia siano nulli e il tensore d'inerzia sia una matrice diagonale. Questi assi sono definiti gli assi principali d'inerzia. Determinare gli assi principali d'inerzia è un problema agli autovalori e autovettori del tensore di inerzia: gli autovettori mostrano le direzioni dei tre assi principali, mentre gli autovalori sono i momenti di inerzia rispetto ai tre assi principali.

Come primo passo è stato necessario determinare il tensore d'inerzia della molecola:

$$\begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \tag{37}$$

I componenti della diagonale della matrice sono i momenti d'inerzia calcolati rispetto gli assi x, y e z:

$$I_{xx} = \sum_{i=1}^{N} m_i(y_i^2 + z_i^2) \tag{38}$$

$$I_{yy} = \sum_{i=1}^{N} m_i(x_i^2 + z_i^2) \tag{39}$$

$$I_{zz} = \sum_{i=1}^{N} m_i(x_i^2 + y_i^2) \tag{40}$$

I componenti della matrice, al di fuori della diagonale, rappresentano i momenti centrifughi di inerzia. Il tensore è sempre una matrice simmetrica per cui $I_{xy} = I_{yx}$, $I_{xz} = I_{zx}$ e $I_{yz} = I_{zy}$.

$$I_{xy} = I_{yx} = \sum_{i=1}^{N} m_i x_i y_i \tag{41}$$

$$I_{xz} = I_{zx} = \sum_{i=1}^{N} m_i x_i z_i \tag{42}$$

$$I_{yz} = I_{zy} = \sum_{i=1}^{N} m_i y_i z_i \tag{43}$$

I momenti e prodotti d'inerzia così definiti fanno riferimento a un corpo rigido costituito da N masse considerate come punti materiali, modello che meglio si adatta ai dati ottenuti tramite le simulazioni *full-atoms* in cui le proprietà degli atomi delle molecole sono determinate in modo puntuale.

Noto il tensore di inerzia è quindi stato possibile ricavare gli autovalori e gli autovettori. Gli autovettori sono stati normalizzati in modo da avere i versori corrispondenti agli assi principali. Durante questa fase si è prestata attenzione all'orientazione dei versori in modo che il loro verso fosse sempre coerente, per ciascun punto del dataset considerato.

Ottenuti i versori è stato possibile costruire la matrice di rotazione, le cui colonne sono i versori corrispondenti agli assi principali di inerzia. Nota la matrice di rotazione è stato possibile quindi ottenere le coordinate, velocità e forze istantanee descritte rispetto alla base che identifica come sistema di riferimento gli assi principali d'inerzia. Questo risultato è stato ottenuto effettuando un semplice prodotto matriciale tra la matrice di rotazione e i vettori posizione, velocità e forza istantanea.

$$[\mathbf{v}]_{a.p.i} = [M]^{a.c.m}_{a.p.i}[\mathbf{v}]_{a.c.m} \tag{44}$$

dove $[\mathbf{v}]_{a.p.i}$ è il vettore descritto rispetto gli assi principali d'inerzia, $[M]^{a.c.m}_{a.p.i}$ è la matrice di cambio di base tra gli assi di riferimento iniziali e gli assi principali d'inerzia, $[\mathbf{v}]_{a.c.m}$ è il vettore di partenza.

Grazie quindi al cambiamento di base appena descritto si è ottenuta l'invarianza alla rotazione del *dataset*.

Il pre-processing dei dati finora descritto, per rendere i dati invarianti alla traslazione e rotazione, è stato eseguito per mezzo di un programma scritto in Python. Lo *script* utilizzato è allegato nell'appendice A.

2.3 Dataset di forze medie

Come già accennato, uno degli obiettivi del presente lavoro è di allenare una rete neurale ad ottenere come predizione le forze medie che agiscono sui singoli atomi della molecola.

Le forze medie, di ogni i-esimo atomo, sono definite dalle seguenti equazioni:

$$\bar{\mathbf{f}}_i = m\bar{\mathbf{a}}_i \tag{45}$$

$$\bar{\mathbf{a}}_i = \frac{\mathbf{v}_i(t+dt) - \mathbf{v}_i(t)}{dt} \tag{46}$$

La forza media $\bar{\mathbf{f}}_i$ corrisponde a un valore mediato nel tempo tra due *frame* non necessariamente immediatamente consecutivi t e $t+dt$. L'accelerazione media $\bar{\mathbf{a}}_i$ è data dalla differenza delle velocità ai *timestep* t e $t+dt$, $\mathbf{v}_i(t+dt)$ e $\mathbf{v}_i(t)$.

Le forze medie così definite non sono valori che si possano estrarre direttamente dalle simulazioni di MD ma sono ottenute a partire dalle velocità, già rese invarianti alla traslazione e rotazione in modo da garantire l'invarianza anche alle forze medie. La creazione di un *dataset* delle forze medie è necessario poiché si vuole utilizzare algoritmi di supervised learning. É quindi necessario fornire *output target* che il modello possa confrontare con le proprie predizioni.

3 Costruzione e applicazione del modello

Nel presente lavoro l'obiettivo, come già accennato, è utilizzare modelli *data-driven*, basati su reti neurali, al fine di predire proprietà quali le coordinate, le forze istantanee e le forze medie agenti sugli atomi delle molecole studiate tramite le simulazioni di MD.

3.1 Costruzione del modello

Per lo sviluppo delle reti neurali si è scelto di utilizzare come linguaggio di programmazione Python e come libreria di riferimento Keras, una libreria open source per l'apprendimento automatico e le reti neurali. Keras è progettata come un'interfaccia a un livello di astrazione superiore di altre librerie simili di più basso livello, e supporta come *back-end* le librerie TensorFlow, Microsoft Cognitive Toolkit e Theano.

Le reti neurali costruite in questo lavoro sono state sviluppate utilizzando come riferimento una classe di reti neurali chiamate CGnets [14], in cui il modello riceve coordinate come *input* e predice un'energia libera. Calcolando poi il gradiente dell'energia rispetto alle coordinate *input* si ottiene una forza.

Siccome l'obiettivo del presente lavoro è quello di predire diverse categorie di *output*, è stato necessario sviluppare differenti modelli che si prestassero meglio al compito a loro assegnato. Essenzialmente, per la predizione delle forze, istantanee o medie, è stato possibile utilizzare la stessa rete neurale, con l'unica differenza che nella fase di *training* i dati *target* forniti alla rete saranno rispettivamente forze istantanee o forze medie. Per predire le coordinate si è invece utilizzato come base il modello creato per le forze, ma sono state effettuate ulteriori modifiche come verrà descritto in seguito.

3.1.1 Modello per la predizione delle forze

Per la costruzione del modello avente come obiettivo la predizione delle forze si è scelto di utilizzare la classe di modelli Keras *Functional* API. La struttura della rete è un *Multi Layer Percetron*, con *input layer*, *hidden layers* e un *output layer*, con la differenza che è stato aggiunto un *lambda layer*, ovvero un *layer* che permette di introdurre espressioni analitiche arbitrarie nel modello.

Il *lambda layer* è stato introdotto in seguito agli *hidden layers*, al fine di calcolare il gradiente dell'energia rispetto alle coordinate *input*. Il gradiente è calcolato grazie alla funzione *gradients* implementata in Keras. L'*output* così ottenuto è quindi trattato come una forza agente sui singoli atomi della proteina considerata [31].
Nel compilare il modello si è scelto di implementare come funzione di errore

(obiettivo di minimizzazione durante il *training*) il *root mean square error* *RMSE*:

$$RMSE \qquad = \sqrt{\sum_{i=1}^{n} \frac{(\hat{y}_i - y_i)^2}{n}} \tag{47}$$

dove $\hat{y}_i$ sono i valori predetti e y_i i valori esatti.

Come ottimizzatore si è implementato l'algoritmo Adam, un'estensione dell'algoritmo della discesa stocastica del gradiente e che può essere usato al suo posto per aggiornare i pesi della rete neurale in modo più efficiente [32].

3.1.2 Modello per la predizione delle coordinate

Come già accennato precedentemente, nel costruire il modello della rete neurale addestrata al fine di predire le coordinate che la proteina assumerà a *timestep* successivi, si è partiti utilizzando come base la struttura della rete che predice una forza, ma si sono effettuate delle modifiche.

La maggiore modifica consiste nell'introduzione di tre ulteriori *lambda layers*, che partono dalle forze per il calcolo rispettivamente dell'accelerazione, della velocità, e infine delle coordinate. La presenza di queste ulteriori espressioni richiede un ulteriore set di dati *input*, che sono le velocità dei singoli atomi. Il *layer input* delle velocità non è collegato agli *hidden layers*, ma al *lambda layer* deputato al calcolo delle velocità, come mostrato nelle seguenti equazioni associate ai tre *lambda layers*.

- *lambda layer*, accelerazione :

$$\mathbf{a}_i = \frac{\mathbf{f}_i}{m_i} \tag{48}$$

dove $\mathbf{f}_i$ è la forza dell'i-esimo atomo della molecola, ricavata dal *lambda layer* gradiente, e m è la massa dell'i-esimo atomo.

- *lambda layer*, velocità :

$$\mathbf{v}_i(t + dt) = \mathbf{v'}_i(t) + \mathbf{a}_i dt \tag{49}$$

dove $\mathbf{v}_i(t + dt)$ è la velocità al *timestep* $(t + dt)$, mentre $\mathbf{v'}_i$ è la velocità al *timestep* di partenza, introdotta come ulteriore *input* della rete. Il valore dt viene scelto in base al *timestep* su cui si vuole effettuare la predizione;

- *lambda layer*, coordinate:

$$\mathbf{x}_i(t+dt) = \mathbf{x'}_i(t) + [\mathbf{v'}_i + \mathbf{v}_i(t+dt)]\frac{dt}{2} \tag{50}$$

dove $\mathbf{x}_i(t+dt)$ sono le coordinate al *timestep* $(t+dt)$, mentre $\mathbf{x'}_i(t)$ sono le coordinate fornite in *input*.

In modo analogo al modello per le forze, la funzione errore implementata è la funzione RMSE e l'ottimizzatore scelto nel compilare il modello è Adam.

3.2 Tuning degli iperparametri

Gli iperparametri sono un insieme di parametri che determinano la dinamica numerica dell'addestramento della rete neurale e la sua struttura, si distinguono per il fatto che non sono parametri che vengono aggiornati durante l'addestramento della rete (come i pesi di ciascun neurone nella rete) ma sono valori che vengono esplicitamente stabiliti dallo sviluppatore del modello.

Il tuning degli iperparametri consiste nella loro ottimizzazione, ancora largamente effettuata manualmente in modo iterativo, al fine di migliorare la *performance*.
Gli iperparametri presi in considerazione in questo lavoro sono:

- *learning rate*, o tasso di apprendimento, iperparametro che controlla la velocità con cui il modello si adatta al problema, in altre parole determina di quanto i pesi della rete devono variare nell'algoritmo di ottimizzazione. Nel presente lavoro si è deciso di utilizzare vari valori di *learning rate* in modo da monitorarne l'effetto sulla *performance*. Il tasso di apprendimento è un valore compreso tra 0 e 1, in questo lavoro di tesi si sono usati i valori: 10^{-6} e 10^{-5}.

- *batch size*, o dimensione del lotto, è definita dal numero di dati che vengono processati dalla rete neurale durante ogni iterazione. La rete può essere addestrata con l'intero *dataset* per ogni iterazione, o può essere invece separatamente allenata con sottogruppi del *dataset*. La *batch size* è stata fissata a 500, ed il suo valore non è stato variato durante il training.

- numero di *epoch*, ovvero il numero di volte in cui l'algoritmo di apprendimento processa l'intero *dataset* durante il *training*. Si è scelto di fissare il numero di *epoch* a 1000000.

- *hidden layers* e numero di neuroni, ovvero il numero di livelli nascosti e il numero di unità che costituiscono i *layers*. Per questi iperparametri

si sono scelte quattro configurazioni (Tabella 1) che sono state alternate in modo sistematico durante i *training*.

Tabella 1: Schema dell'architettura degli *hidden layers*. x: *layer* presente. -: *layer* assente.

Architettura *hidden layers*				
Architettura	I *layer*	II *layer*	III *layer*	IV *layer*
	1000 neuroni	500 neuroni	250 neuroni	125 neuroni
1	x	-	-	-
2	x	x	-	-
3	x	x	x	-
4	x	x	x	x

- funzioni di attivazione, si è scelto di utilizzare come funzione di attivazioni per ogni *hidden layer* la funzione ReLU.

3.3 Training del modello

Il *training* di una rete neurale supervisionata (*supervised learning*), in cui sia gli *input* che gli *output* di un set di *training* sono noti, consiste nel confronto tra il valore predetto dalla rete ed il corrispondente *output* noto, detto valore *target*. In funzione dell'errore così calcolato, i parametri (pesi) della rete vengono modificati al fine di minimizzare l'errore. Per un *tuning* ottimale degli iperparametri si è scelto di svolgere differenti *training* dei vari modelli in modo da potere stabilire quali valori degli iperparametri risultassero in un migliore risultato predittivo della rete.

Durante la fase di *training* si vuole evitare il fenomeno dell'*overfitting*, ovvero dell'eccessivo adattamento, a seguito del quale si ottengono predizioni molto accurate per i dati utilizzati nella fase di addestramento, ma predizioni pessime nel momento in cui la rete deve generalizzare su dati nuovi. Per evitare l'*overfitting*, ed avere quindi una reale capacità predittiva, durante il *training* una parte del *dataset* viene riservata per quello che è definito *validation set*, ovvero un set di dati utilizzati non per l'addestramento, ma per la validazione dei risultati ottenuti con il *training set*. Il *validation set* quindi fornisce una valutazione sulla *perfomance* del modello addestrato. Nei vari training si è scelto di assegnare il 10% del *dataset* come *validation set*[33].

Nei modelli si sono implementate delle *Callbacks*, ovvero set di funzioni che sono applicate durante precisi momenti della procedura di *training*, e che consentono di ottenere informazioni sullo stato del modello e controllare il

processo di addestramento della rete durante il *training* [34]. Le *Callback* utilizzate nei modelli sviluppati sono:

- *EarlyStopping*, ovvero una funzione di blocco del codice che interrompe il *training* quando un valore monitorato smette di migliorare. La metrica monitorata in questo lavoro è la *val loss*, ossia il valore della funzione errore (RMSE) per il *validation set*. Lo scopo di questa *callback* è di evitare il fenomeno di *overfitting*, limitando il numero di *epoch*.

- *ModelCheckpoint*, funzione che salva il modello e i pesi del modello in certi intervalli predefiniti, o quando il modello migliora (*best model*) rispetto alla metrica monitorata. Si è scelto di salvare solo il *best model*, e la metrica monitorata anche in questo caso è stata la *val loss*.

I *training* dei modelli sono stati effettuati grazie alla funzione *model.fit* di Keras, in cui sono passati come parametri le varie opzioni, appena illustrate, che si è scelto di imporre [4].

Caso studio $(\mathbf{AAQAA})_\mathbf{3}$ Il *dataset* disponibile per la proteina $(AAQAA)_3$ comprende 10^5 *frame*, per i quali sono noti coordinate, velocità e forze istantanee. Nell'utilizzare *dataset* di dimensione inferiore a quello disponibile, si è scelto *frame* consecutivi a partire dal primo disponibile. Per le forze medie si è scelto di considerare il valore medio tra due *frames* consecutivi, originando così un totale di 99999 campioni. Si è deciso di predire le coordinate al *timestep* successivo rispetto alle coordinate *input*. I *training* effettuati per questo caso studio sono illustrati nella Tabella 2.

Tabella 2: Schema dei *training* effettuati con il *dataset* $(AAQAA)_3$. Per ogni training, sono specificati i valori degli iperparametri considerati.

Training $(AAQAA)_3$			
Modello	*Dataset* (# frame)	Architettura	*Learning rate*
Output: forze istantanee	100000 10000 5000	1,2,3,4 1,2,3,4 1,2,3,4	10^{-6}, 10^{-5} 10^{-6}, 10^{-5} 10^{-6}, 10^{-5}
Output: forze medie	99999	1,2,3,4	10^{-6}, 10^{-5}
Output: coordinate	99999	1,2,3,4	10^{-6}, 10^{-5}

Caso studio Dialanina Il *dataset* considerato per la proteina dialanina comprende 49999 *frame*, per i quali sono noti coordinate, velocità e forze istantanee. Nell'utilizzare *dataset* di dimensione inferiore a quello disponibile, si è scelto *frame* consecutivi a partire dal primo disponibile. Per le forze medie si è scelto di considerare il valore medio tra due *frame* consecutivi, originando così un totale di 49998 campioni. Si è deciso di predire le coordinate al *timestep* successivo rispetto alle coordinate *input*. I *training* effettuati sono illustrati nella Tabella 3.

Tabella 3: Schema dei *training* effettuati con il *dataset* dialanina. Per ogni training, sono specificati i valori degli iperparametri considerati.

Training Dialanina			
Modello	*Dataset* (# frame)	Architettura	*Learning rate*
Output: forze istantanee	49999	1,2,3,4	10^{-6}, 10^{-5}
	25000	1,2,3,4	10^{-6}, 10^{-5}
	5000	1,2,3,4	10^{-6}, 10^{-5}
Output: forze medie	49998	1,2,3,4	10^{-6}, 10^{-5}
Output: coordinate	49998	1,2,3,4	10^{-6}, 10^{-5}

3.4 Testing del modello

Il *test set* è un set di dati forniti alla rete per una valutazione finale imparziale del modello addestrato con il *training set*. Idealmente i dati scelti per il *testing* del modello dovrebbero essere dati non utilizzati per il *training* del modello, in modo da garantire un'effettiva imparzialità nel valutare l'errore che la rete compie nelle sue predizioni [35].

Nel presente lavoro si è scelto di utilizzare come *test set* i campioni di dati non utilizzati nel *training*; prendendo per esempio un *training set* di 10000 *frames* del caso studio $(AAQAA)_3$ in cui il dataset disponibile è di 100000, il *test set* sarà composto dai restanti 90000 campioni di dati. Eccezionalmente, nel caso in cui si è deciso di utilizzare l'intero *dataset* per il *training set* e *validation set*, si è reso necessario utilizzare il *validation set* anche come *test set*.

Il *test set* è stato fornito come *input* ai modelli salvati durante i vari allenamenti delle reti (*best models*). I *testing* dei modelli è effettuato grazie alla funzione Keras *model.evaluate* e le predizioni sono state fornite dalla funzione *model.predict* [4].

Nelle appendici B,C,D si riportano i seguenti *script* utilizzati nel presente lavoro:

Allegato B, sviluppo del modello della rete neurale che predice la forza e suo *training*;

Allegato C, sviluppo del modello della rete neurale che predice le coordinate e suo *training*;

Allegato D, *testing* dei modelli.

4 Risultati

In questo capitolo si illustreranno e analizzeranno le prestazioni delle reti neurali per la predizione delle forze istantanee, medie e delle coordinate. I modelli sviluppati al fine di predire queste proprietà sono stati applicati per entrambi i *dataset* ottenuti dalle simulazioni di dinamica molecolare, $(AAQAA)_3$ e dialanina, monitorandone le *performance* e analizzando i valori predetti.

4.1 Curve di apprendimento

Una curva di apprendimento è il grafico della *perfomance* di un modello di apprendimento in funzione dell'esperienza o del tempo. Le curve di apprendimento sono uno strumento largamente utilizzato per gli algoritmi che imparano ovvero, ottimizzano i parametri interni gradualmente nel tempo, come le reti neurali utilizzate in questo lavoro, per valutare la *performance* del modello. La metrica utilizzata per valutare l'apprendimento tendenzialmente è una metrica che deve essere minimizzata, come una funzione *loss*, o errore, dove valori inferiori indicano un apprendimento migliore. Valutare la funzione *loss* sul *training set* permette di analizzare se il modello sia in grado di imparare una relazione tra *input* e *target output*, mentre una valutazione sul *validation set* offre una idea delle capacità di generalizzazione del modello, ovvero consente di capire se il modello durante la fase di allenamento va incontro al fenomeno di *overfitting* [36].

La forma e la dinamica di una curva di apprendimento può essere quindi utilizzata per analizzare il comportamento di un modello di *machine learning*. Le più comuni dinamiche che si possono osservare sono:

- *underfit*, si verifica quando il modello non è in grado di ottenere un valore sufficientemente basso della funzione *loss* sul *training set*. Un modello in stato di *underfit* può quindi essere riconosciuto anche solo dalla curva di apprendimento della funzione *loss* sul *training set*. La curva si presenta come una linea piatta o come rumore, indicando che la rete non è stata in grado di imparare dal *training set*[37][36].

- *overfit*, in questo caso il modello ha imparato troppo bene il *training set* e non è in grado di generalizzare. Questa problematica si può valutare come già precedentemente accennato sul *validation set*. La curva di apprendimento che indica questo fenomeno presenta quindi una grafico della *train loss* che continua a scendere mentre la *validation loss* decresce solo per un periodo e poi inizia di nuovo a crescere [33][36].

- *good fit*, si caratterizza per essere l'obiettivo dei modelli, una via di mezzo tra *overfit* e *underfit*, nel quale il modello impara dal *training set* senza perdere però la capacità di generalizzare. Entrambe le curve, *training* e *validation* decrescono fino a un punto di stabilità con un gap, minimo, tra i due valori finali di *loss* [36].

Nel presente lavoro di tesi si è scelto di monitorare le *performance* dei modelli durante i *training* con le curve di apprendimento, usando come metrica di controllo il valore *loss* in funzione dell'aumentare del numero di *epoch*.

4.1.1 Caso studio $(AAQAA)_3$

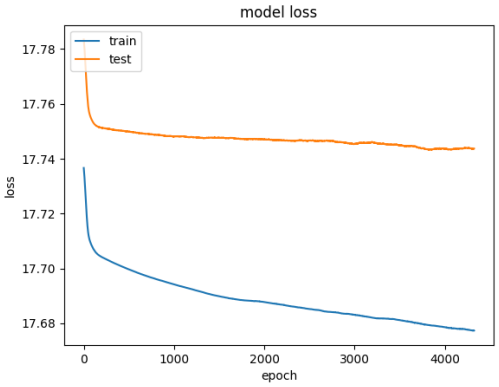
Modello forze istantanee Le curve di apprendimento dei *training* delle reti neurali aventi come *output* le forze istantanee evidenziano differenti dinamiche al variare della dimensione del *dataset* di *input* fornito al modello, e al variare degli iperparametri.

- *Dataset* 100000 (Figure 13 e 14), le curve mostrano essenzialmente delle dinamiche di *good fit*, a prescindere degli iperparametri scelti, con l'eccezione dei casi con architettura 3 o 4 con *learning rate* 10^{-5}, in cui è possibile notare dinamiche di *overfit*.

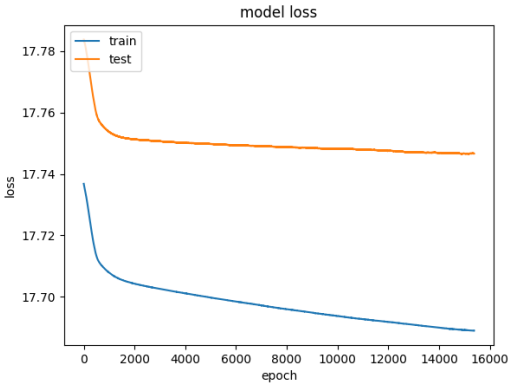
-*Dataset* 10000 (Figure 15 e 16), al diminuire della dimensione del *dataset* si nota un aumentare dei fenomeni di *overfitting*, per un valore di learning rate di 10^{-5}. Il numero di *layer* sembra non influenzare l'andamento delle curve di apprendimento.

-*Dataset* 5000 (Figure 17 e 18), il comportamento delle curve peggiora, nuovamente, ad eccezione per l'architettura della rete con un singolo *hidden layer* (17a). In particolare, le curve dei modelli con *learning rate* 10^{-5} mostrano comportamenti di *overfitting*. Inoltre, per il caso (18b) la rete va incontro a un caso di *underfit*.

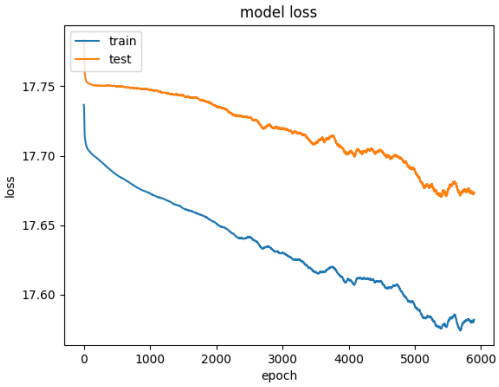
La dimensione del *dataset* fornito alla rete neurale è un aspetto fondamentale, al diminuire del volume di dati forniti in *input* infatti le *perfomance* peggiorano. Grazie alle curve di apprendimento è stato possibile evidenziare per il *training* del modello come il valore di *learning rate* 10^{-5} comporti problemi sul fronte dell'*overfitting*. L'architettura della rete, ovvero il numero di *hidden layers* non influenza in maniera determinante la *performance* del modello.



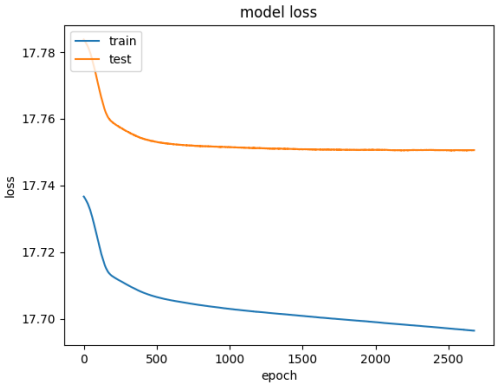
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}

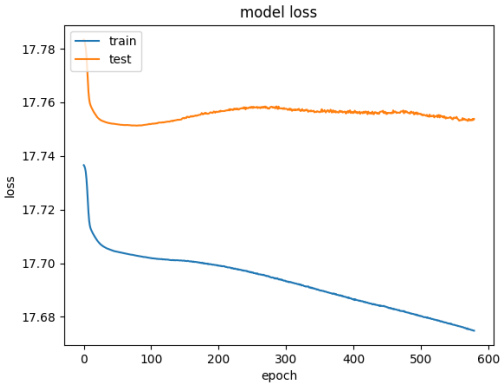


(c) Architettura 2, LR = 10^{-5}

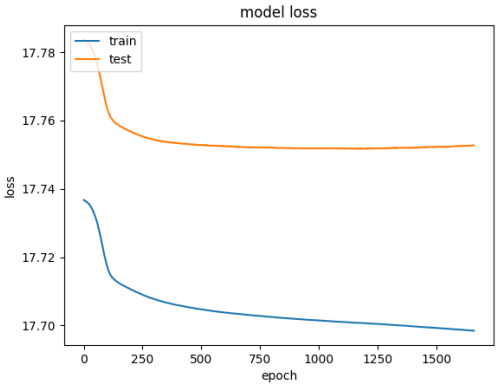


(d) Architettura 2, LR = 10^{-6}

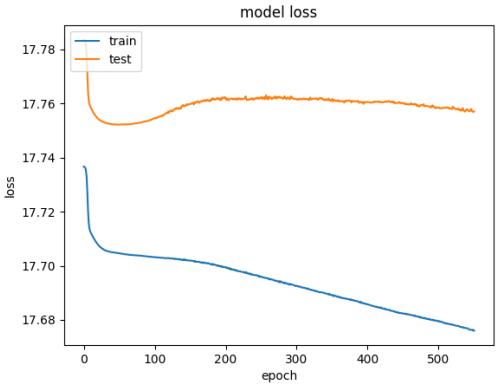
Figura 13: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* di $(AAQAA)_3$ con 100000 input.



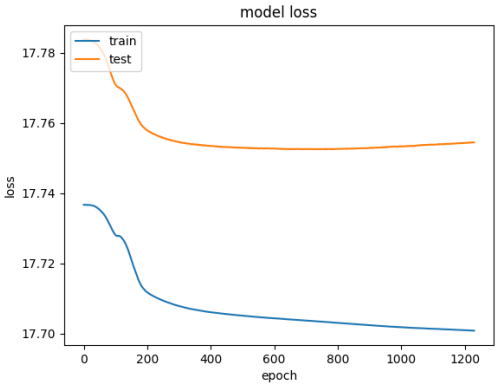
(a) Architettura 3, LR = 10^{-5}



(b) Architettura 3, LR = 10^{-6}

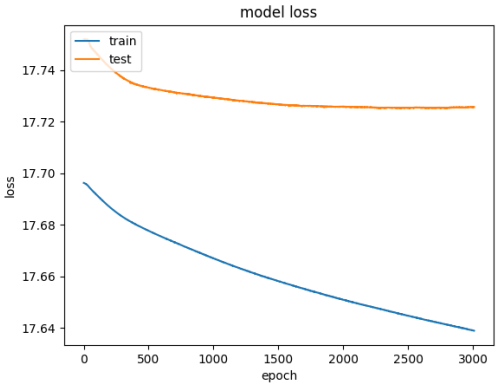


(c) Architettura 4, LR = 10^{-5}

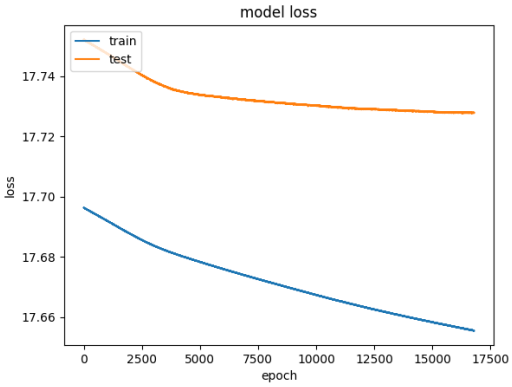


(d) Architettura 4, LR = 10^{-6}

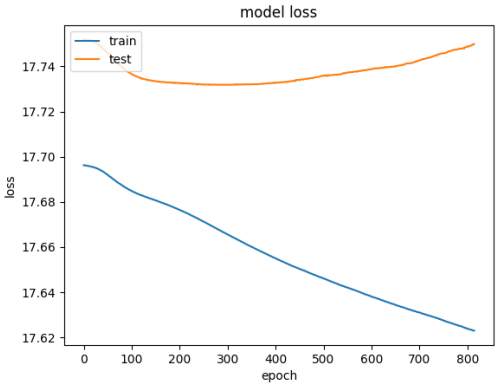
Figura 14: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* di $(AAQAA)_3$ con 100000 input.



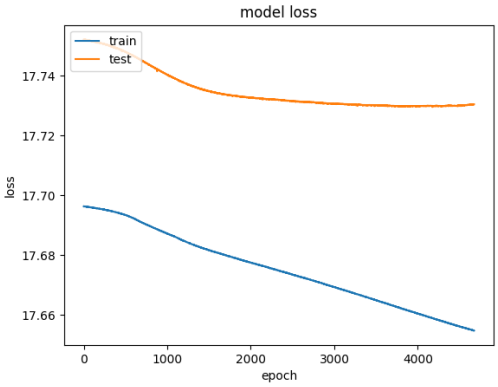
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}



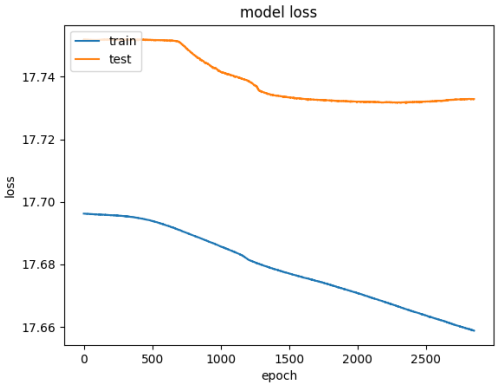
(c) Architettura 2, LR = 10^{-5}



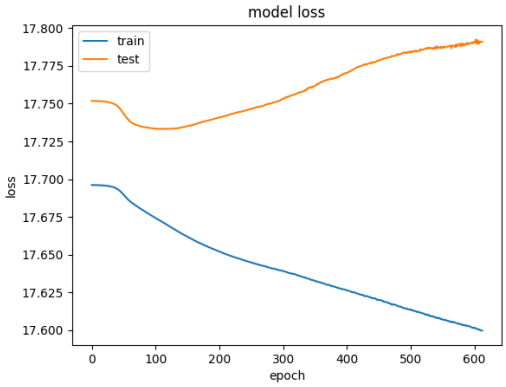
(d) Architettura 2, LR = 10^{-6}

Figura 15: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* di $(AAQAA)_3$ con 10000.

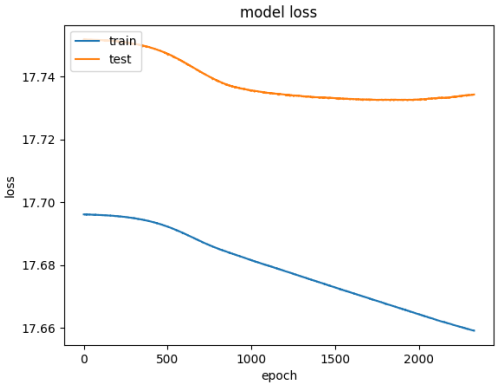
(a) Architettura 3, LR = 10^{-5}



(b) Architettura 3, LR = 10^{-6}

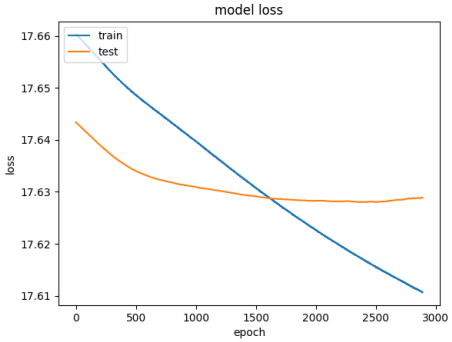


(c) Architettura 4, LR = 10^{-5}

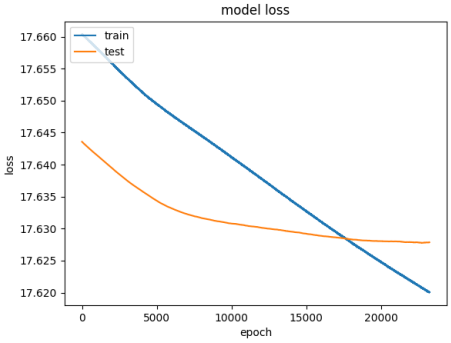


(d) Architettura 4, LR = 10^{-6}

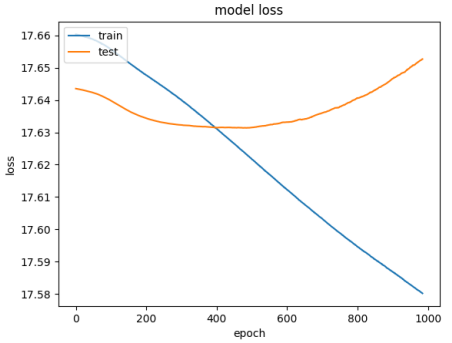
Figura 16: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* di $(AAQAA)_3$ con 10000 input.



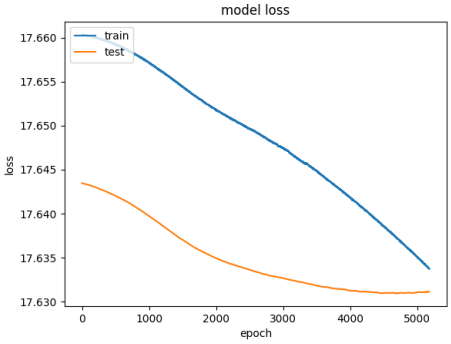
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}

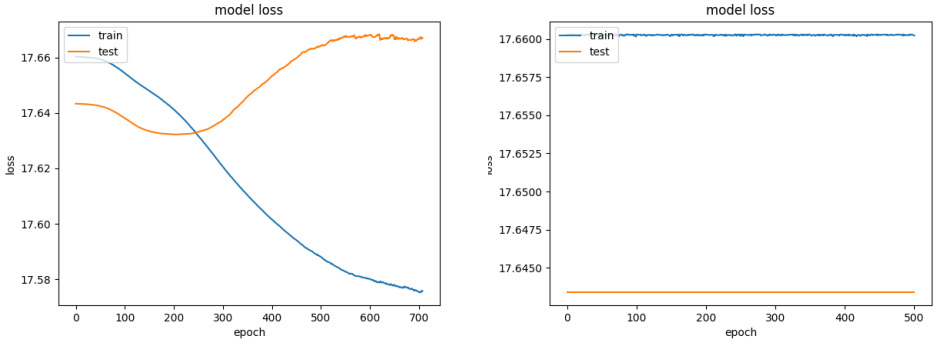


(c) Architettura 2, LR = 10^{-5}

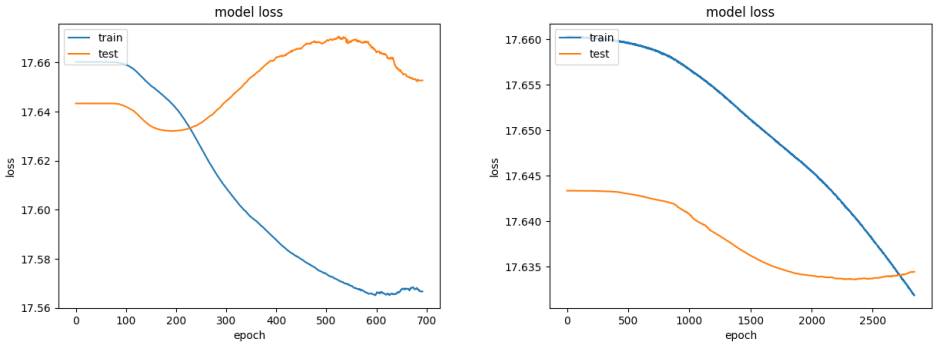


(d) Architettura 2, LR = 10^{-6}

Figura 17: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* di $(AAQAA)_3$ con 5000 input.



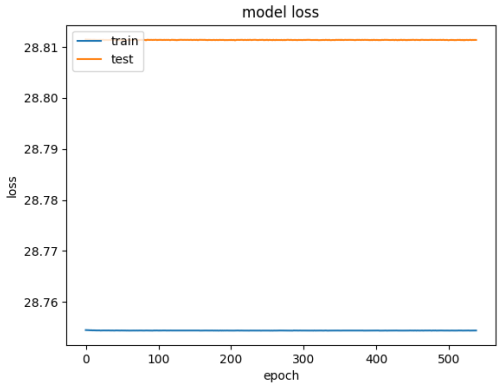
(a) Architettura 3, LR = 10^{-5} (b) Architettura 3, LR = 10^{-6}



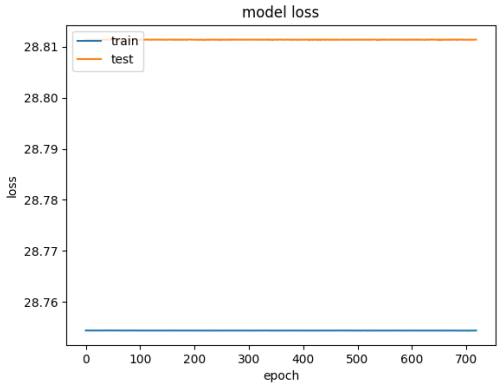
(c) Architettura 4, LR = 10^{-5} (d) Architettura 4, LR = 10^{-6}

Figura 18: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* di $(AAQAA)_3$ con 5000 input.

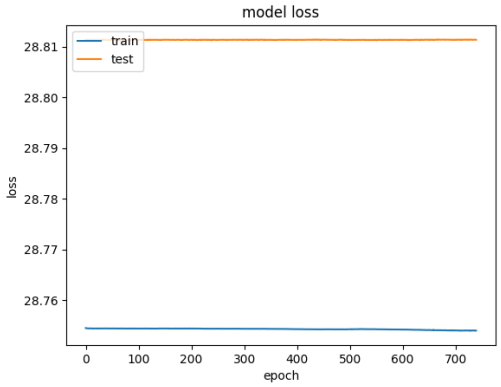
Modello forze medie Le curve di apprendimento (Figure 19 e 20), nel caso del *training* delle reti neurali aventi come *output* le forze medie, mostrano come a prescindere degli iperparametri le reti non siano in grado di imparare dagli *input*. Le curve mostrano, infatti andamenti costanti tipici del fenomeno di *underfit*. Si è scelto di limitare i *training* sull'intero *dataset* disponibile in questo caso, viste le pessime *performance* del modello.



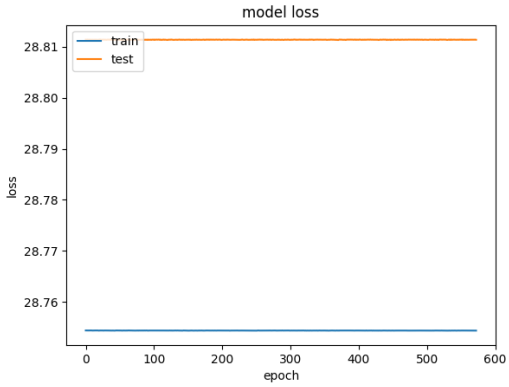
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}

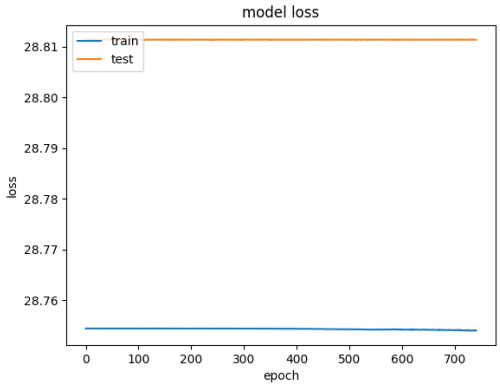


(c) Architettura 2, LR = 10^{-5}

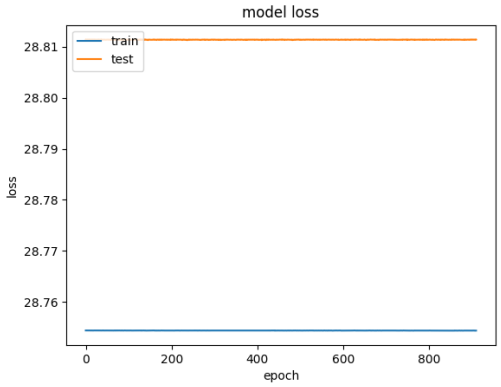


(d) Architettura 2, LR = 10^{-6}

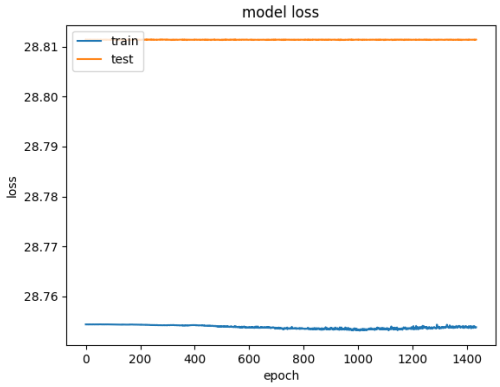
Figura 19: Curve di apprendimento del modello avente come output le forze medie, applicato al *dataset* $(AAQAA)_3$ con 99999 input.



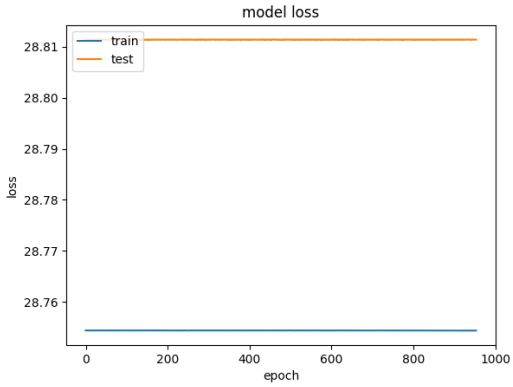
(a) Architettura 3, LR = 10^{-5}



(b) Architettura 3, LR = 10^{-6}



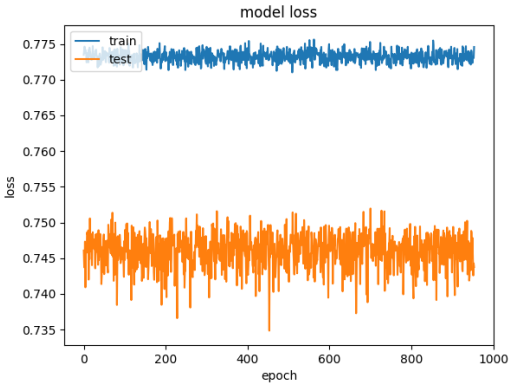
(c) Architettura 4, LR = 10^{-5}



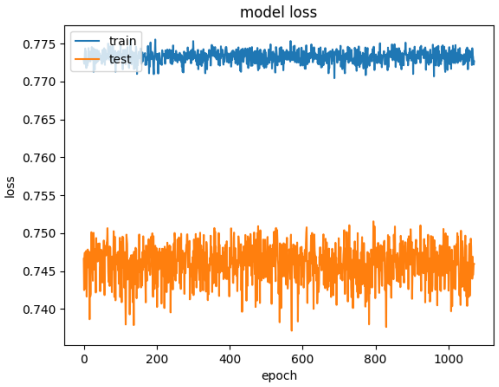
(d) Architettura 4, LR = 10^{-6}

Figura 20: Curve di apprendimento del modello avente come output le forze medie, applicato al *dataset* $(AAQAA)_3$ 99999 input.

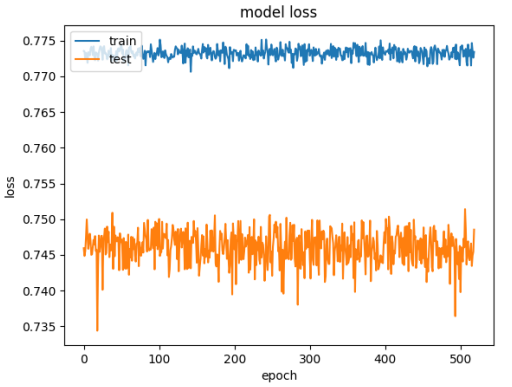
Modello coordinate Le curve di apprendimento (Figure 21 e 22) dei *training* delle reti neurali che hanno l'obiettivo di predire le coordinate, a un *timestep* successivo, mostrano delle dinamiche di *underfit* che si evincono dall'andamento rumoroso delle curve. In modo analogo al caso del modello delle forze medie, si sono limitati i *training* sull'intero *dataset*, a causa delle scarse capacità del modello di apprendere.



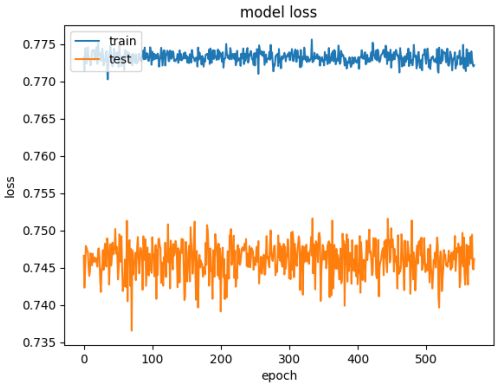
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}

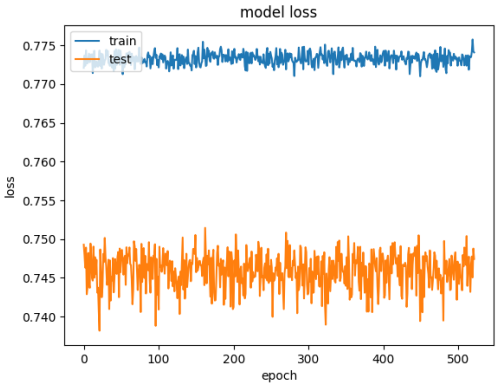


(c) Architettura 2, LR = 10^{-5}

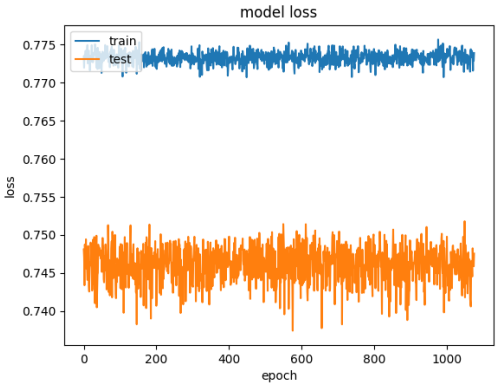


(d) Architettura 2, LR = 10^{-6}

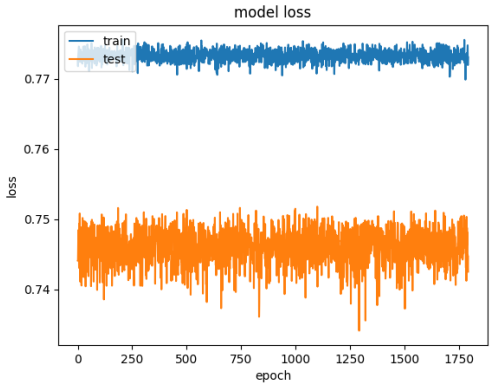
Figura 21: Curve di apprendimento del modello avente come output le coordinate, applicato al *dataset* $(AAQAA)_3$ con 99999 input.



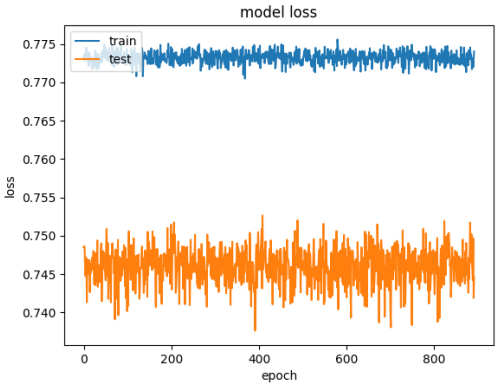
(a) Architettura 3, LR = 10^{-5}



(b) Architettura 3, LR = 10^{-6}



(c) Architettura 4, LR = 10^{-5}



(d) Architettura 4, LR = 10^{-6}

Figura 22: Curve di apprendimento del modello avente come output le coordinate, applicato al *dataset input* $(AAQAA)_3$ con 99999 input.

4.1.2 Caso studio Dialanina

Modello forze istantanee Nelle curve di apprendimento dei *training* delle reti neurali aventi come *output* le forze istantanee è possibile vedere differenti dinamiche in base agli iperparametri e alle dimensioni del *dataset*.

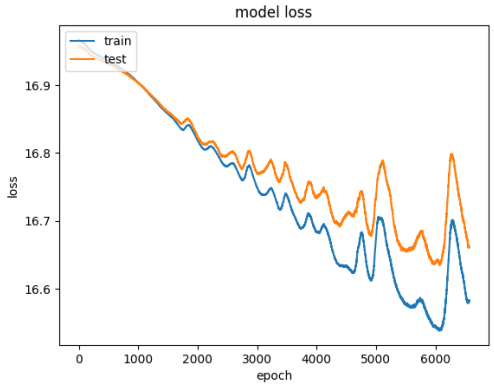
- *Dataset* 49999 (Figure 23 e 24), le curve presentano delle dinamiche particolari che non rientrano perfettamente nelle tre categorie illustrate, ma in generale si evince dalle curve un andamento della *loss* sul *validation set* anomalo che si ipotizza possa coincidere con un *overfitting* della rete. Il training 23b fa eccezione, le curve di apprendimento

mostrano un discreto andamento decrescente per entrambi i valori di *loss* e quindi una *performance* migliore del modello.

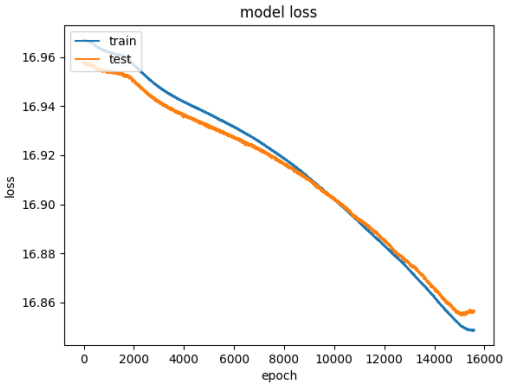
-*Dataset* 25000 (Figure 25 e 26), le curve di apprendimento dell'architettura 1, per entrambi i valori di *learning rate* della rete mostrano una dinamica decrescente per entrambi i valori di *loss*, mentre per i restanti *training* le curve sono quasi piatte (*underfit*).

-*Dataset* 5000 (Figure 27 e 28), analogamente a quanto avviene per il *dataset* 25000, le curve sono piatte con le seguenti eccezioni: l'architettura 1 con *learning rate* 10^{-5} (27a) in cui si osserva un leggero andamento decrescente, l'architettura 4 con *learning rate* 10^{-5} (28c) in cui la curva di apprendimento mostra un leggero fenomeno di *overfitting*.

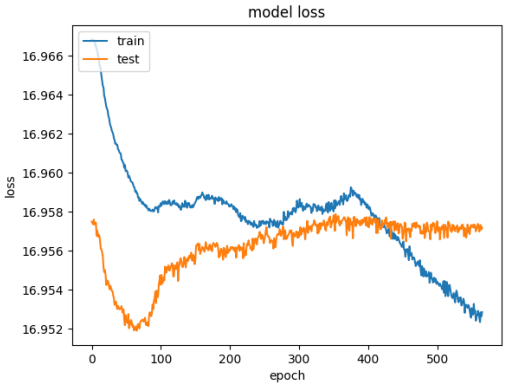
Diversamente dal caso studio $(AAQAA)_3$ non è facilmente intuibile un *trend* tra gli iperparametri scelti e la qualità delle *perfomance* delle reti. Tuttavia, tendenzialmente, i risultati migliori sono stati ottenuti con le reti neurali aventi architettura 1. La dimensione del *dataset* incide sul *training* del modello ma non è stato evidenziato un netto miglioramento delle prestazioni all'aumentare della dimensione come nel caso $(AAQAA)_3$.



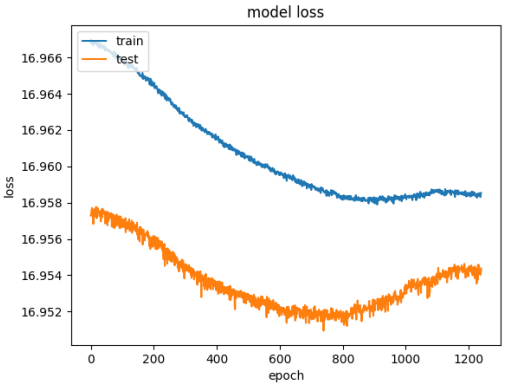
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}

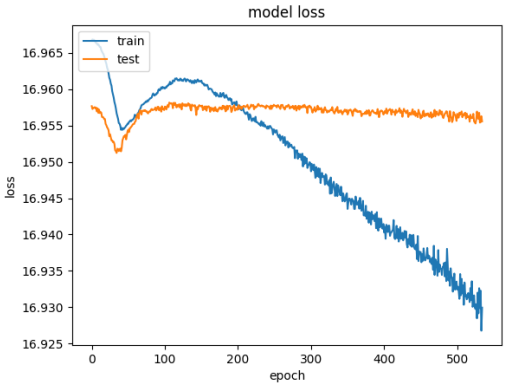


(c) Architettura 2, LR = 10^{-5}

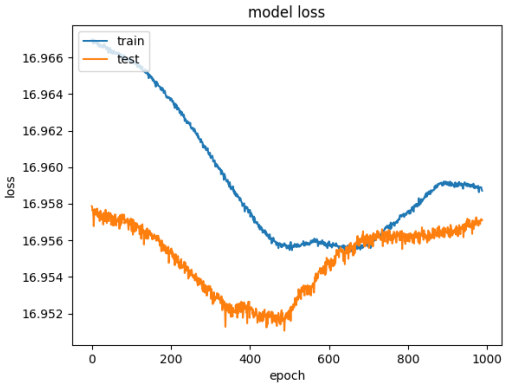


(d) Architettura 2, LR = 10^{-6}

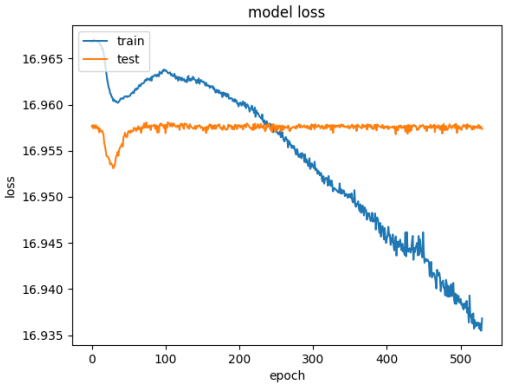
Figura 23: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* dialanina con 49999 input.



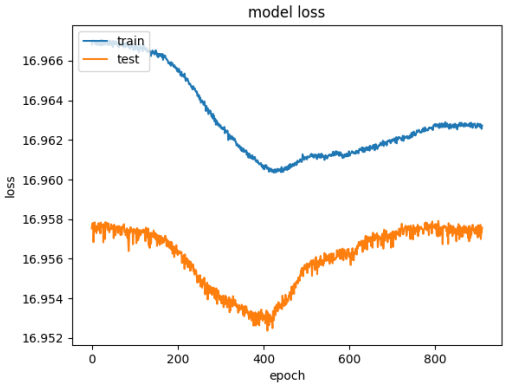
(a) Architettura 3, LR = 10^{-5}



(b) Architettura 3, LR = 10^{-6}

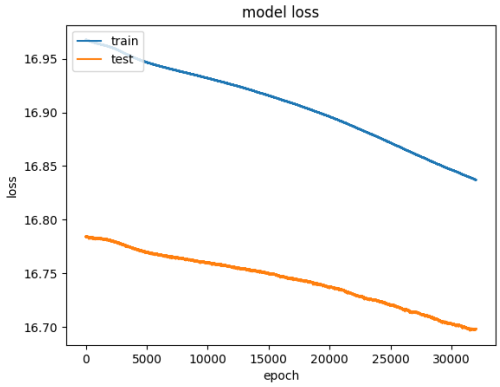


(c) Architettura 4, LR = 10^{-5}

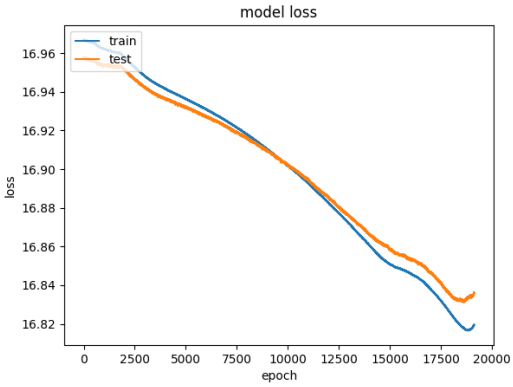


(d) Architettura 4, LR = 10^{-6}

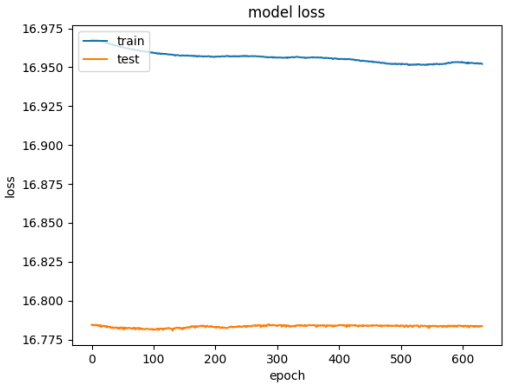
Figura 24: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* dialanina con 49999 input.



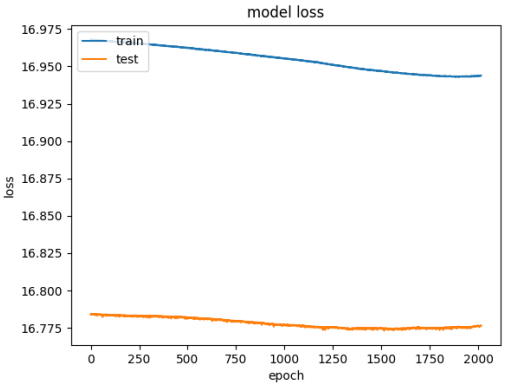
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}

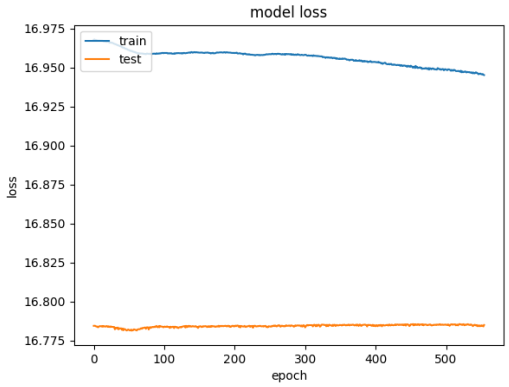


(c) Architettura 2, LR = 10^{-5}

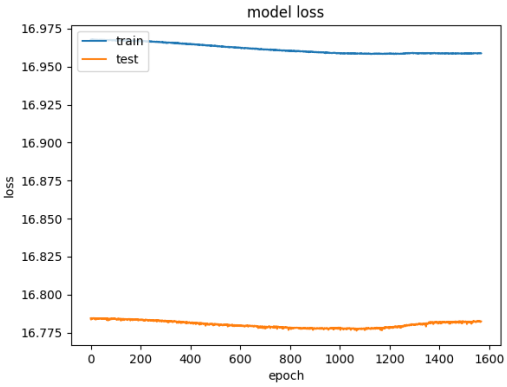


(d) Architettura 2, LR = 10^{-6}

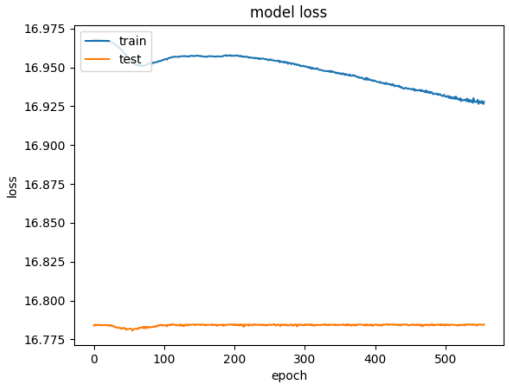
Figura 25: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* dialanina con 25000 input.



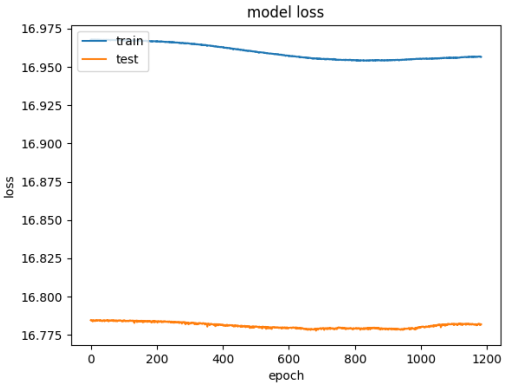
(a) Architettura 3, LR = 10^{-5}



(b) Architettura 3, LR = 10^{-6}

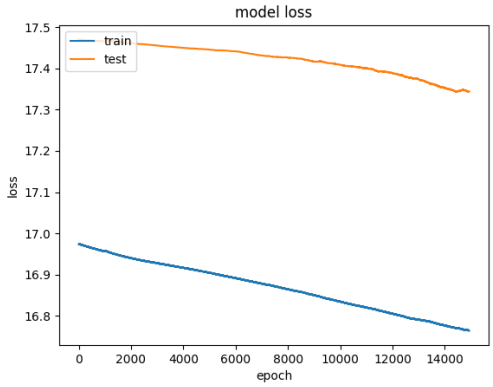


(c) Architettura 4, LR = 10^{-5}

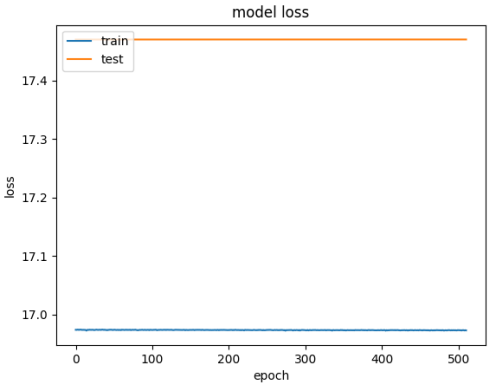


(d) Architettura 4, LR = 10^{-6}

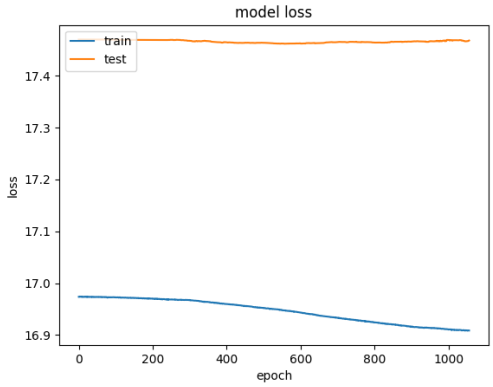
Figura 26: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* dialanina con 25000 input.



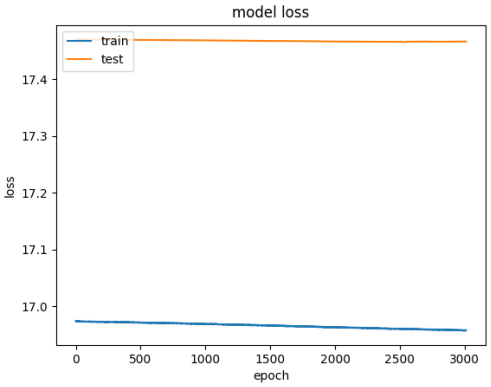
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}

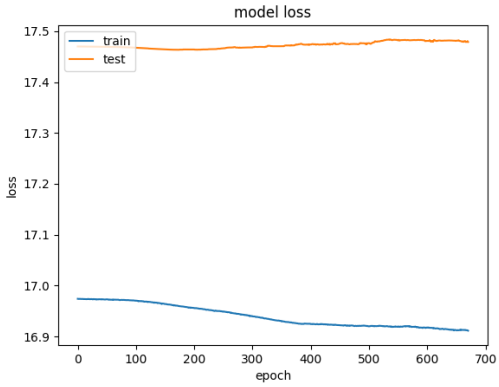


(c) Architettura 2, LR = 10^{-5}

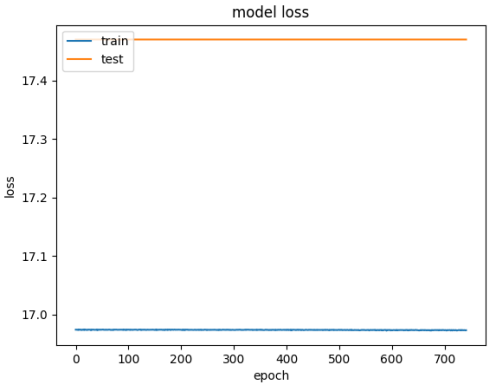


(d) Architettura 2, LR = 10^{-6}

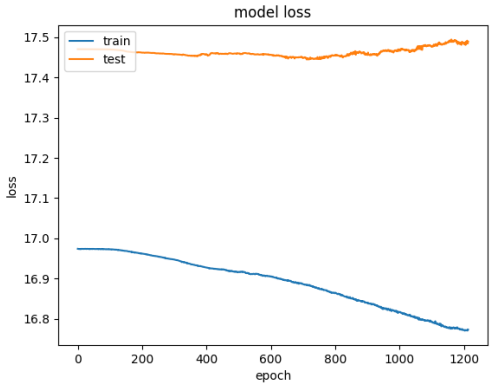
Figura 27: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* dialanina con 5000 input.



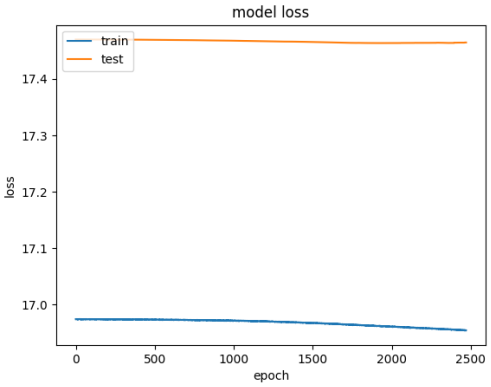
(a) Architettura 3, LR = 10^{-5}



(b) Architettura 3, LR = 10^{-6}



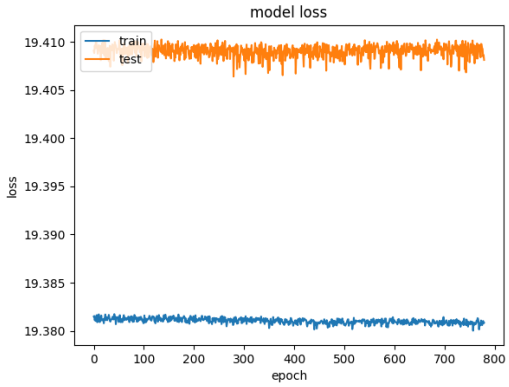
(c) Architettura 4, LR = 10^{-5}



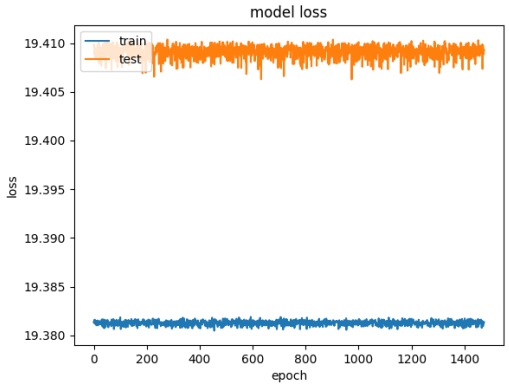
(d) Architettura 4, LR = 10^{-6}

Figura 28: Curve di apprendimento del modello avente come output le forze istantanee, applicato al *dataset* dialanina con 5000 input.

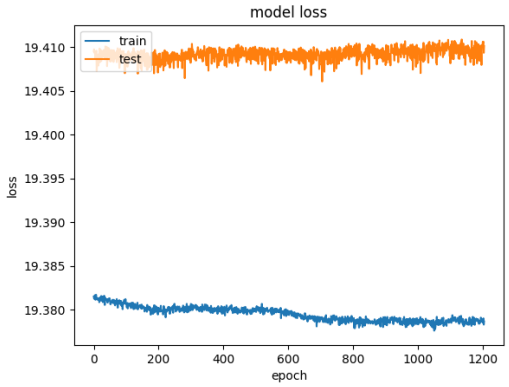
Modello forze medie Le curve di apprendimento (Figure 29 e 30) evidenziano una comune dinamica rumorosa dei grafici: il modello, a prescindere degli iperparametri utilizzati, non è in grado di imparare dal *dataset*. Date le scarse *perfomance* si è scelto di limitare i *training* anche per questo modello sull'intero *dataset*.



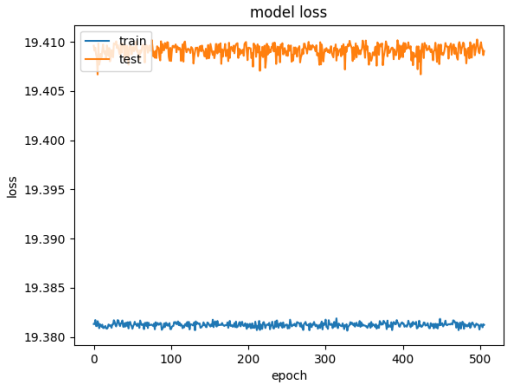
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}

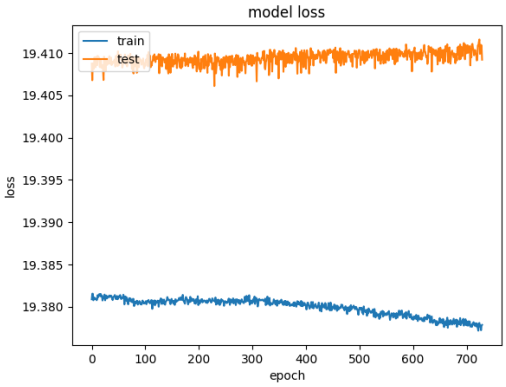


(c) Architettura 2, LR = 10^{-5}

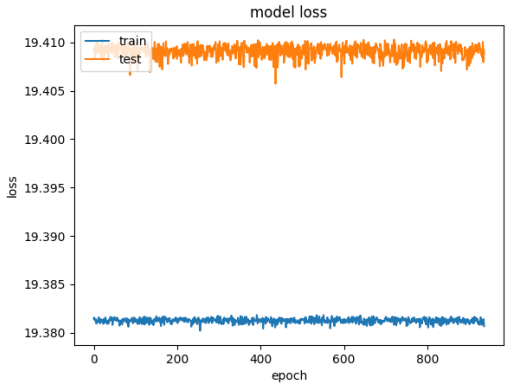


(d) Architettura 2, LR = 10^{-6}

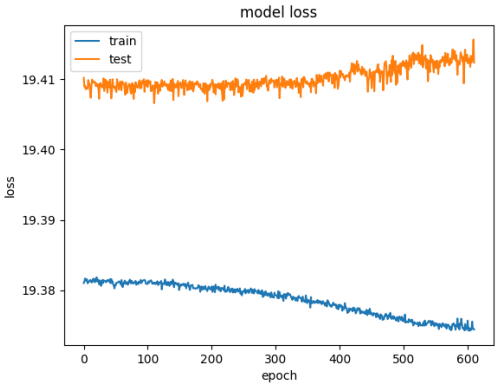
Figura 29: Curve di apprendimento del modello avente come output le forze medie, applicato al *dataset* dialanina con 49998 input.



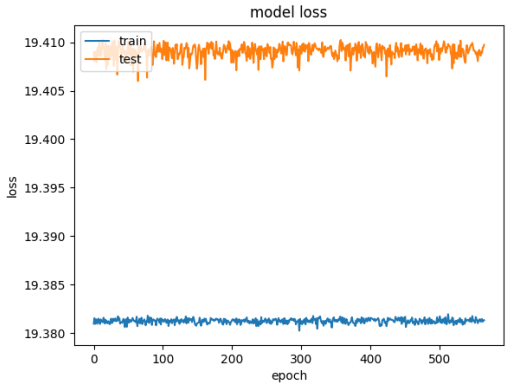
(a) Architettura 3, LR = 10^{-5}



(b) Architettura 3, LR = 10^{-6}



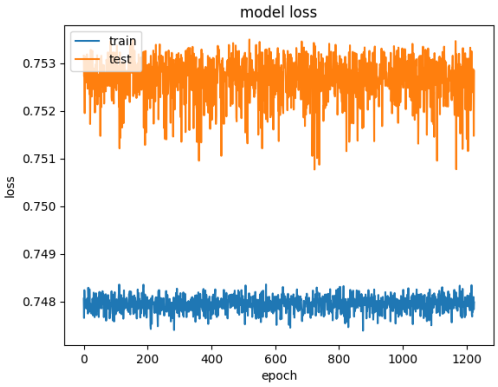
(c) Architettura 4, LR = 10^{-5}



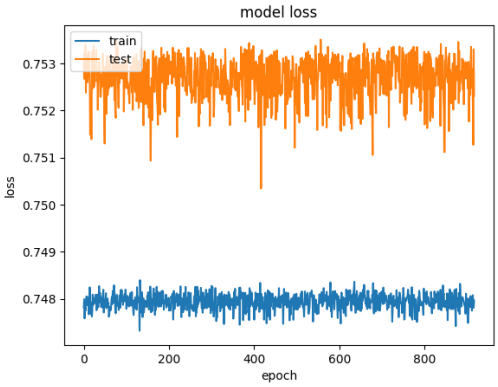
(d) Architettura 4, LR = 10^{-6}

Figura 30: Curve di apprendimento del modello avente come output le forze medie, applicato al *dataset* dialanina con 49998 .

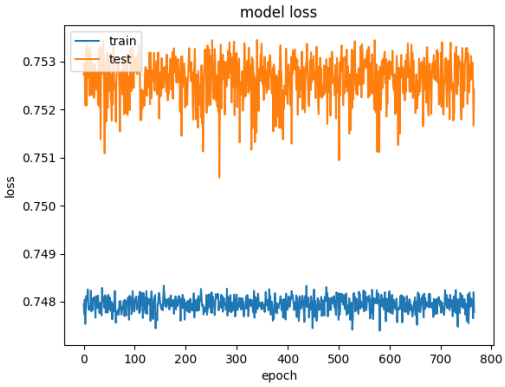
Modello coordinate Le curve di apprendimento (Figure 31 e 32) mostrano un andamento rumoroso, il modello non è in grado di imparare dal *dataset*. Anche in questo caso si è scelto di utilizzare solamente l'intero volume di dati per i *training*.



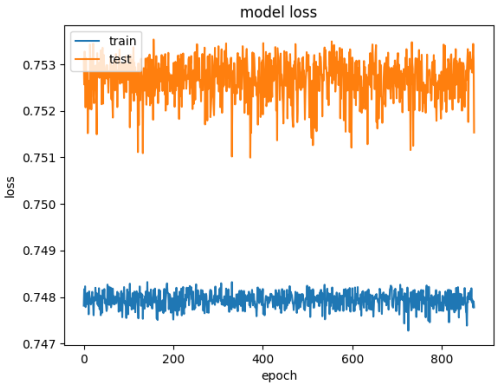
(a) Architettura 1, LR = 10^{-5}



(b) Architettura 1, LR = 10^{-6}

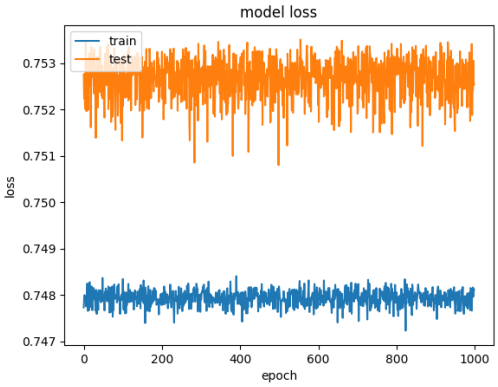


(c) Architettura 2, LR = 10^{-5}

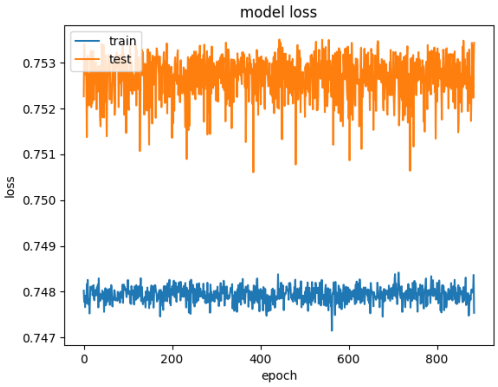


(d) Architettura 2, LR = 10^{-6}

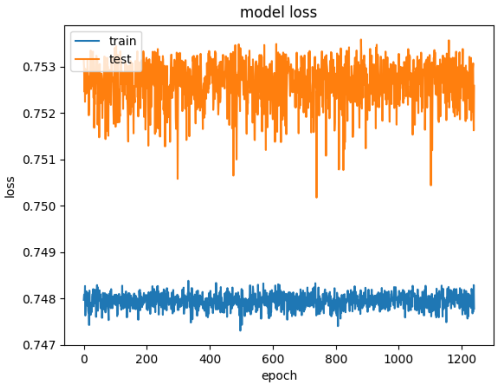
Figura 31: Curve di apprendimento del modello avente come output le coordinate, applicato al *dataset* dialanina con 49998 input.



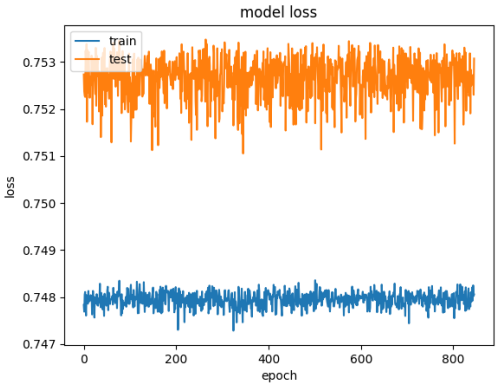
(a) Architettura 3, LR = 10^{-5}



(b) Architettura 3, LR = 10^{-6}



(c) Architettura 4, LR = 10^{-5}



(d) Architettura 4, LR = 10^{-6}

Figura 32: Curve di apprendimento del modello avente come output le coordinate, applicato al *dataset* dialanina con 49998 input.

Le curve di apprendimento hanno permesso di evidenziare come l'unico modello, con delle *perfomance* accettabili, sia quello che predice le forze istantanee, con risultati migliori per il *dataset* $(AAQAA)_3$. Le curve hanno evidenziato un altro particolare fenomeno, riscontrato in vari *training*: il valore della *validation loss* è spesso inferiore rispetto alla *train loss* (Figure 17, 18, 21-26). Quest'ultimo è un fenomeno molto particolare siccome per come è definito il *validation set* l'errore dovrebbe essere maggiore rispetto a quello sul *training set*. Le cause che possono essere alla base di questo problema sono: 1) un possibile *data-leakage* tra il *training set* e il *validation set*, ovvero i dati del *training set* sono molto simili a quelli del *validation set*; 2) il *validation set* è più 'facile' da imparare rispetto al *training set*, e questo

può verificarsi quando il *validation set* è troppo piccolo o non è propriamente campionato.

4.2 Confronto tra valori predetti e valori reali

Se mediante le curve di apprendimento è stato possibile monitorare le *performance* dei modelli sul *training set* e sul *validation set*, si è in aggiunta voluto analizzare la qualità delle singole predizioni rispetto a valori reali di forze istantanee, medie e coordinate ricavate grazie alle simulazioni di dinamica molecolare. Durante il *testing* delle reti neurali si è scelto di salvare i dati predetti, ed a partire da essi si sono ottenute le coordinate degli atomi della molecola, da potere confrontare con le coordinate reali. Il confronto è stato effettuato a livello visivo tramite il programma di visualizzazione molecolare VMD, che ha permesso di rappresentare visivamente la struttura molecolare delle molecole studiate.

4.2.1 Considerazioni sulla qualità delle predizioni

Nel caso delle reti neurali con *output* coordinate un confronto tra valori predetti e valori reali è immediatamente possibile e non richiede nessun *post-processing* dei dati ottenuti dai modelli. Diversamente, i modelli che restituiscono come *output* delle forze, istantanee o medie, richiedono dei passaggi intermedi. A partire delle forze si devono infatti ottenere delle coordinate, essenzialmente replicando le espressioni implementate con gli aggiuntivi *lambda layers* nel modello delle coordinate. In particolare, si è utilizzata l'equazione 48 per ricavare l'accelerazione, l'equazione 49 per le velocità e l'equazione 50 per le coordinate, inserendo in v_i' e x_i' i dati estratti dal *dataset*.

Siccome nel caso dei modelli delle forze si utilizzano dei valori intermedi (per appunto, le forze predette dalla rete) per arrivare a quelle che si considerano delle coordinate predette, si è voluto fare delle considerazioni sull'impatto che le forze di partenza hanno sul risultato finale. Si è scelto di fare un esperimento assumendo per assurdo che le forze predette dalla rete neurale siano dei valori *random* compresi tra i valori di massimo e minimo delle forze, rese invarianti, estratte dalla simulazione di MD. Ovvero si è voluto considerare l'ipotesi che il modello non impari a riconoscere la relazione tra le coordinate *input* e i valori *target*, e restituisca dei valori casuali. Per questa dimostrazione si è scelto di considerare il caso studio $(AAQAA)_3$ e di utilizzare i dati corrispondenti al *timestep* 1. Generata la matrice delle forze random che agiscono sui 180 atomi della molecola si sono ottenute le coordinate al *timestep* successivo con le equazioni precedentemente citate. Le coordinate così ottenute, matrice di dimensione [180,3], sono state utilizzate

per creare un file nel format pdb (*Protein Data Bank*), un tipo di file che descrive le strutture tridimensionali delle proteine contenute nel corrispondente *database*. L'utilizzo di questo tipo di formattazione delle coordinate è necessario per l'utilizzo del programma VMD. Il risultato di questa prova evidenzia come partendo da valori casuali di forza non si ottenga la struttura della proteina $(AAQAA)_3$, ma una struttura incompleta (Figura 33), in cui certi legami non sono riconosciuti dal programma di visualizzazione poiché non corrispondono a legami possibili a livello fisico.

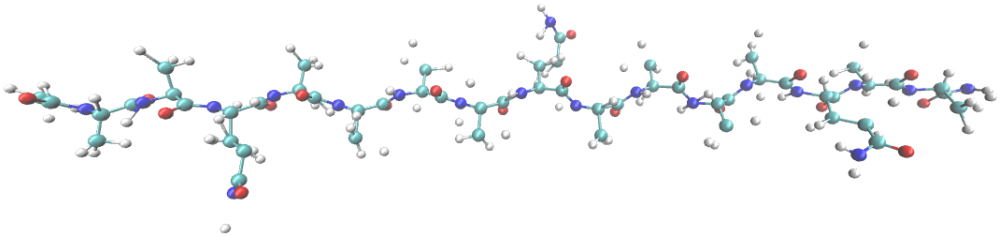
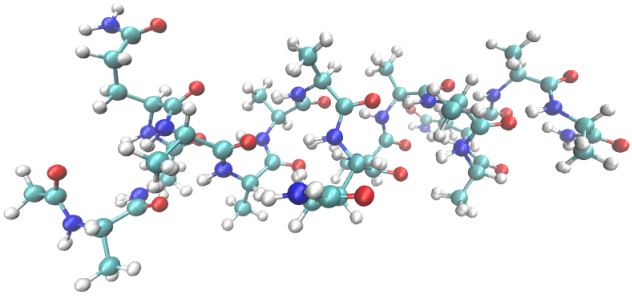


Figura 33: Struttura molecolare ottenuta con valori random delle forze, si osservano degli atomi di idrogeno non collegati da legami alla struttura principale.

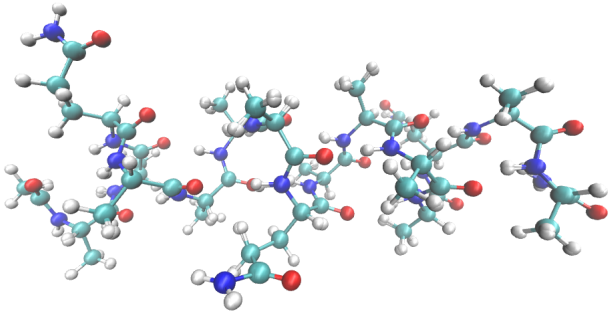
4.2.2 Caso studio $(AAQAA)_3$

Per il confronto visivo si è scelto di utilizzare i valori predetti dai modelli che hanno evidenziato le *perfomance* migliori e il valore predetto corrispondente al *frame* 95001, che corrisponde alle coordinate della proteina al tempo 9.5 ns.

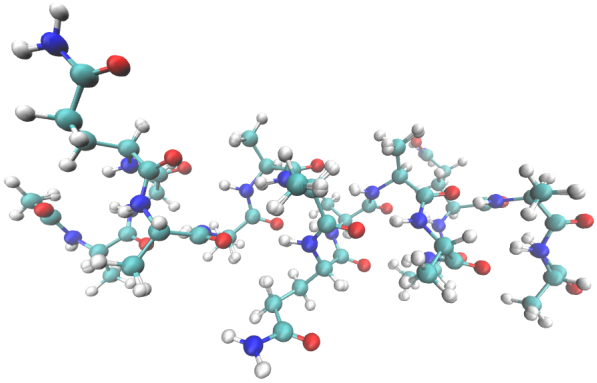
Modello forze istantanee Le strutture ottenute grazie ai valori di forza istantanea coincidono largamente con la struttura reale della proteina al *frame* 95001, i maggiori scostamenti sono individuabili nelle posizioni degli atomi di idrogeno (Figura 34).



(a) *Dataset* 5000, Ar. 1, LR = 10^{-5}



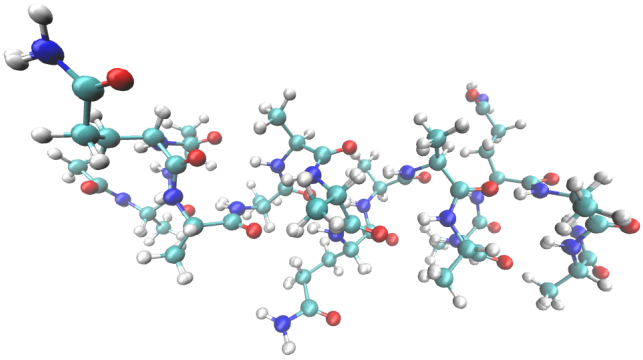
(b) *Dataset* 10000, Ar. 1, LR = 10^{-5}



(c) *Dataset* 100000, Ar. 1, LR = 10^{-5}

Figura 34: Sovrapposizione della struttura reale della proteina $(AAQAA)_3$ alla struttura ottenuta a partire dai valori di forze istantanee predetti dal modello.

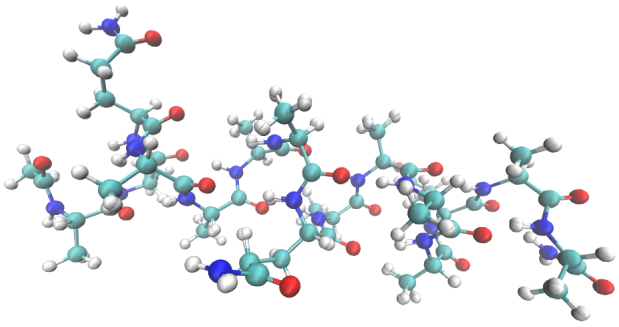
Modello forze medie Le due strutture della proteina, reale e predetta, coincidono in larga parte, il risultato si dimostra interessante in quanto le curve di apprendimento di questo modello evidenziano come la rete non sembri in grado di imparare (Figura 35).



(a) *Dataset* 99999, Ar. 1, LR = 10^{-5}

Figura 35: Sovrapposizione della struttura reale della proteina $(AAQAA)_3$ alla struttura ottenuta a partire dai valori di forze medie predetti dal modello.

Modello coordinate Analogamente al caso del modello delle forze medie, nonostante le *performance* della rete neurale siano pessime, le coordinate predette rispecchiano una struttura similare a quella *target* (Figura 36).



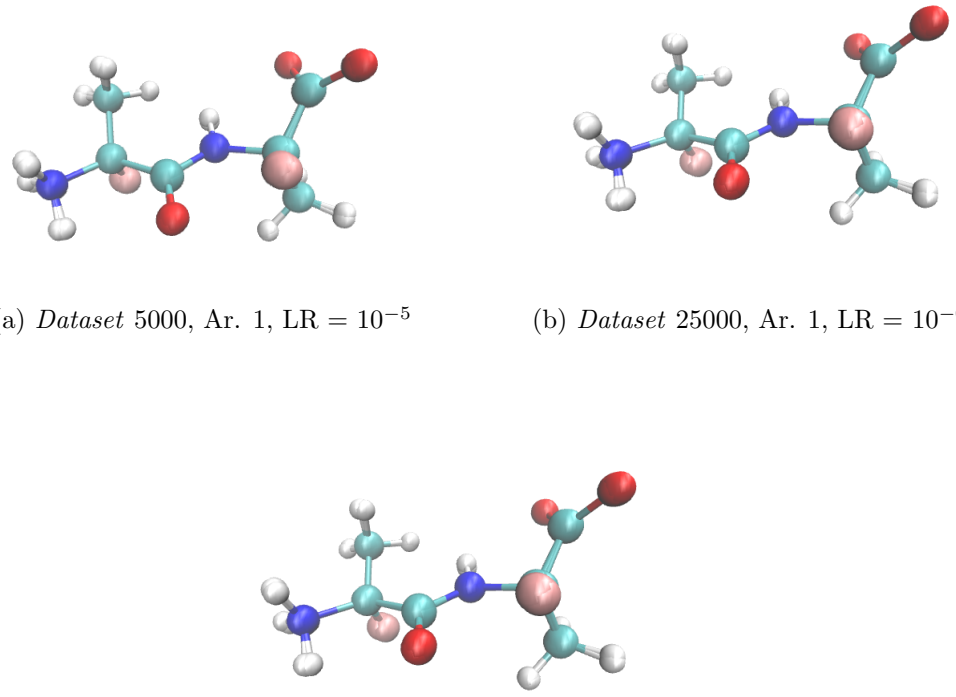
(a) *Dataset* 99999, Ar. 1, LR = 10^{-5}

Figura 36: Sovrapposizione della struttura reale della proteina $(AAQAA)_3$ alla struttura predetta dal modello avente come output le coordinate.

4.2.3 Caso studio dialanina

Per il confronto visivo si è scelto di utilizzare i valori predetti dai modelli che hanno evidenziato le *perfomance* migliori e il valore predetto corrispondente al *frame* 47501 che corrisponde alle coordinate della proteina al tempo 104.75 ns.

Modello forze istantanee Le strutture ottenute grazie ai valori di forza istantanea coincidono quasi totalmente con la struttura reale della proteina al frame 47501, oltre a deviazioni degli atomi di idrogeno si notano delle lievi discrepanze tra gli atomi di carbonio (Figura 37).



(a) *Dataset* 5000, Ar. 1, LR = 10^{-5}

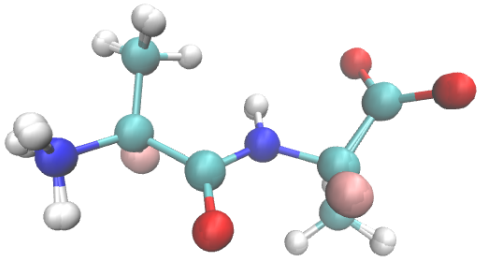
(b) *Dataset* 25000, Ar. 1, LR = 10^{-6}

(c) *Dataset* 49999, Ar. 1, LR = 10^{-6}

Figura 37: Sovrapposizione della struttura reale della proteina dialanina alla struttura ottenuta a partire dai valori di forze istantanee predetti dal modello.

Modello forze medie Similarmente alla proteina $(AAQAA)_3$, anche se le curve di apprendimento del modello mostrano una incapacità da parte della

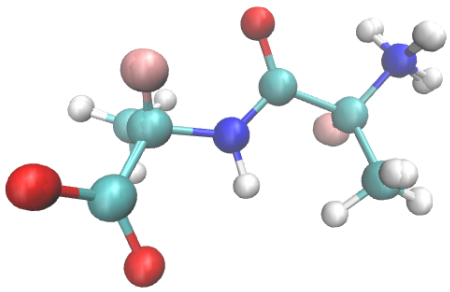
rete di apprendere dagli *input*, la struttura predetta non si discosta di molto da quella reale (Figura 38).



(a) *Dataset* 49998, Ar. 2, LR = 10^{-5}

Figura 38: Sovrapposizione della struttura reale della proteina dialanina alla struttura ottenuta a partire dai valori di forze medie predetti dal modello.

Modello coordinate Anche in questo caso, nonostante le scarse *perfomance* del modello, le coordinate predette dalla rete rispecchiano discretamente la struttura reale della proteina (Figura 39).



(a) *Dataset* 49998, Ar. 1, LR = 10^{-5}

Figura 39: Sovrapposizione della struttura reale della proteina dialanina alla struttura predetta dal modello avente come output le coordinate.

Nell'appendice E è allegato lo script Python utilizzato per ottenere un file pdb a partire dalle coordinate predette dalla rete neurale.

5 Confronto con algoritmi presenti in letteratura

L'applicazione di algoritmi di machine learning nell'ambito di simulazioni di dinamica molecolare è un campo di ricerca in continua espansione. In letteratura è possibile trovare molti lavori al riguardo che si concentrano, come già accennato precedentemente, su singoli aspetti delle simulazioni di MD.

Per questo lavoro di tesi è stato di grande interesse il lavoro presentato da Wang et al. (2019) [14], in cui si è utilizzata una classe di reti neurali chiamate CGnets per determinare il *force field* di un sistema *Coarse-Grained* (CG). La modellazione *Coarse-Grained* di un sistema complesso consiste nel proporre una rappresentazione semplificata, in cui più atomi del sistema sono uniti a formare singole entità dette *beads*, del sistema di partenza. Le CGnets sono modelli in grado di imparare funzioni di energia libera di un sistema *Coarse-Grained* e possono essere allenate per predire le forze agenti sul sistema considerato (Figura 40). Questa classe di reti inoltre garantisce le invarianze rilevanti a livello fisico e permette di incorporare a priori delle conoscenze fisiche per evitare di predire strutture non realistiche a livello fisico.

L'obiettivo finale del lavoro di Wang et al. era determinare profili di energia libera di sistemi CG e confrontarli con i profili ottenuti partendo da simulazioni di dinamica molecolare *all-atom* di due proteine modello in acqua: dialanina e chignolina.

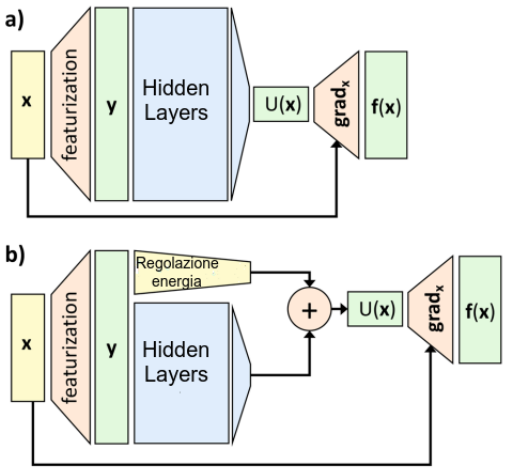


Figura 40: Schema delle rete neurali proposte in [14]. a) CGnet. b) CGnet con regolarizzazione energetica. Figura tratta da [14], con modifiche.

La capacità delle CGnets di fornire come *output* le forze agenti sul sistema studiato è stata la fonte di ispirazione per il *building* la costruzione delle reti neurali utilizzate nel presente lavoro di tesi. Tuttavia, l'approccio ai dati ottenuti dalle simulazioni MD e l'obiettivo finale del lavoro sono differenti. Infatti, se nel presente lavoro si è scelto di utilizzare come *input* le coordinate cartesiane rese invarianti con una fase di *pre-processing*, nelle CGnet si è implementato un *feature layer* (Figura 40) che trasforma le coordinate cartesiane in variabili interne alla rete, invarianti alla traslazione e rotazione, come angoli, distanze tra coppie di atomi o *beads* nel caso di modelli CG e angoli diedri (un processo indicato come *featurization*).

Inoltre, in una delle possibili varianti di CGnet (Figura 40b) è stato scelto di introdurre un potenziale armonico su angoli e distanze di legame, che consente di effettuare una regolarizzazione energetica del sistema e impedisce a simulazioni con CGnets di divergere in stati non fisicamente possibili.

Un aspetto presente in CGnets e conservato in questo lavoro è l'utilizzo di un *layer* per calcolare il gradiente dell'energia potenziale ($U(\mathbf{x})$), *output* degli *hidden layers*, rispetto alle coordinate cartesiane $\mathbf{x}$ al fine di ottenere le forze ($\mathbf{f}(\mathbf{x})$).

5.1 Dialanina Coarse Grained: *Supervised learning* tramite CGnet

L'algoritmo utilizzato da Wang et al. [14] è disponibile su GitHub (https://github.com/coarse-graining/cgnet) attraverso un tutorial in cui è possibile replicare il *training* di una rete CGnet (schema di Figura 40b) fornendo come *input* le coordinate della dialanina (coarse-granata in 5 *beads*) e come valori *target* le forze. Il *dataset* fornito a tale scopo è una versione troncata, 10000 *frames*, del dataset realmente impiegato nelle pubblicazioni su CGnet [14][38].

Oltre ad eseguire questo tutorial con i dati appositamente forniti si è scelto di provare a fare delle leggere modifiche al codice CGnet: si sono cioè fornite alla rete le coordinate e le forze, *all-atom*, ottenute in questo lavoro (l'intero *dataset* ottenuto in solvente implicito per la dialanina). Ciò è stato possibile poiché il modello CG utilizzato in CGnet non contiene il solvente, e quindi le procedure di *coarse-graining* devono imparare l'energia libera di solvatazione in modo analogo alla simulazioni svolte nel presente lavoro di tesi.

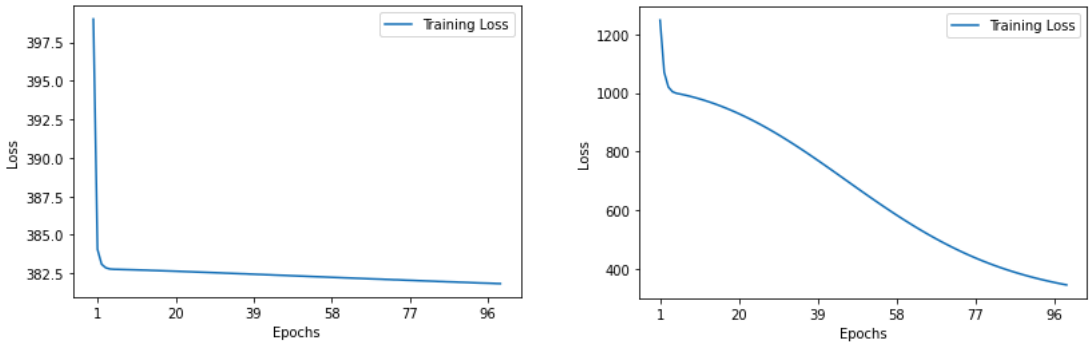
Durante i *training* di entrambi i *dataset* quello originariamente utilizzato per lo sviluppo di CGnet, e quello ottenuto in questo lavoro si è monitorato il valore della *train loss*. La funzione *loss* utilizzata nel tutorial è la funzione

MSE (Mean Squared Error) che è il quadrato della funzione RMSE (utilizzata invece dai nostri modelli). Si è visto che a parità di numero di *epoch* i valori di *loss* finale sono rispettivamente 381.83 per il *dataset* originale e 346.67 per il *dataset* generato in questo lavoro (Figura 41). Il modello seppure con delle modifiche sembra quindi essere in grado di gestire come *input* anche le coordinate di ogni singolo atomo della proteina.

Un'osservazione interessante che si può fare sui valori di *loss* è che essi sono simili, indipendentemente dal dataset di partenza considerato. Inoltre, i valori di *loss* ottenuti tramite CGnet non si discostano molto da quelli mostrati nel Capitolo 4 e generati attraverso i nostri codici ML, come si può osservare convertendo la *loss* espressa tramite MSE in RMSE. Questa comparazione è possibile se si tiene in conto che

$$(train\,loss)^{CGnet}_{RMSE} = \sqrt{(train\,loss)^{CGnet}_{MSE}} = \sqrt{346.67} = 18.62 \tag{51}$$

La $(train\,loss)_{RMSE}$ ottenuta nei *training* riportati nel Capitolo 4 sul *dataset* relativo alla dialanina varia in un range di valori compresi tra 16.975 e 16.80, in linea con quanto ottenuto tramite CGnet.



(a) Dataset orinale CGnet

(b) Dataset generato in questo lavoro

Figura 41: Andamento della *train* loss all'aumentare del numero di *epoch*, per il dataset originariamente utilizzato in CGnet (a), e quello generato in questo lavoro (b).

6 Conclusioni

Nel presente lavoro di tesi si è scelto di accoppiare i risultati di simulazioni di dinamica molecolare con modelli di *machine-learning* basati su reti neurali. Nello specifico si sono utilizzate le reti neurali per predire proprietà legate all'evoluzione temporale di conformazioni proteiche, quali le coordinate degli atomi dopo un certo numero di passi di integrazione, oppure le forze istantanee e le forze medie agenti sulle singole conformazioni.

Le simulazioni MD hanno permesso di soddisfare la necessità di modelli *data-driven*, come le reti neurali, di avere un elevato numero di dati da processare. In questo lavoro si è rivolta l'attenzione a due peptidi modello, simulati in solvente implicito: $(AAQAA)_3$ e dialanina. Siccome si è voluta garantire l'invarianza alla traslazione e alla rotazione, è stato necessario manipolare i dati estratti dalle simulazioni a questo fine, calcolando la posizione del centro di massa, il tensore di inerzia e la matrice di cambio base associata agli assi principali di inerzia.

Parte del lavoro svolto in questa tesi è stato quindi rivolto allo sviluppo e formulazione di modelli di reti neurali che fossero capaci di predire le proprietà stabilite come obiettivi. Come base di riferimento si sono utilizzate la classe di rete neurali *Multi Layer Perceptron* e le CGnets, classi di reti sviluppate da Wang et al. [14]. Le proprietà che si è cercato di predire, come già affermato, sono tre: coordinate, forze istantanee e medie, e per ognuna di essa si è modellata una rete neurale.

Grazie alle curve di apprendimento è stato possibile monitorare le prestazioni delle rete neurali durante i *training*, ed ottimizzare gli iperparametri. I risultati hanno evidenziato come il modello utilizzato per le predizioni delle forze istantanee sia l'unico in cui è possibile notare una capacità di apprendimento dal *dataset* fornito come *input*. Si è notato inoltre come il modello si adatti meglio al caso della proteina $(AAQAA)_3$ rispetto alla dialanina; la trasferibilità del modello risulta essere limitata. In questa direzione, approcci più generali, affrontati anche in letteratura, rivolgono l'attenzione su come i dati vengono forniti alla rete e sull'implementazione di strutture più complesse che permettano ai modelli stessi di estrarre descrittori più accurati delle molecole in input.

Le curve hanno evidenziato inoltre durante alcuni *training* un fenomeno comune ad entrambi i casi studio: il valore di *validation loss* maggiore al valore di *train loss*. Le cause di questo fenomeno risiedono essenzialmente in un errato campionamento del *validation set* e del *training set*. Il *validation set* scelto troppo piccolo, o troppo facile da imparare, o vi potrebbe essere un *data-leakage* tra *training set* e *validation set*. Un *data-leakage* avviene quando i dati compresi in entrambi i *set* sono troppo simili tra di loro. Una soluzione

a questo problema potrebbe essere modificare i *dataset* ottenuti tramite le simulazioni MD, per esempio riducendo la frequenza di campionamento ed aumentando il numero di *timestep* tra un *frame* ed il successivo.

Oltre a valutare le *performance* delle reti neurali si è voluta valutare la qualità delle singoli predizioni rispetto a valori reali. A questo scopo, durante la fase di validazione dei modelli, si sono salvati i valori predetti dalla rete, che sono stati utilizzati per ottenere una corrispondente conformazione proteica. Quest'ultima è stata poi confrontata con la struttura reale estratta dalle simulazioni. Tramite questo confronto è stato possibile notare che anche le reti neurali aventi come *output* forze medie e coordinate sono comunque in grado di restituire valori che rispecchiano strutture realistiche delle proteine. Quindi i modelli sviluppati per la predizione delle forze medie e delle coordinate sono comunque un buon punto di partenza, nonostante richiedano ancora del lavoro.

Il presente lavoro di tesi si presenta inoltre come possibile punto di partenza per un utilizzo *online* delle reti neurali, ovvero l'utilizzo delle reti in parallelo alle simulazioni di MD, al fine di sostituire gli algoritmi di MD in alcune iterazioni, e quindi limitare i costi computazionali legati alla determinazione delle forze agenti sul sistema.

Appendice A

```
## RENDERE LE POSIZIONI E LE VELOCITÀ INVARIANTI
## RISPETTO ALLA TRASLAZIONE E ROTAZIONE
## ESEMPIO CASO STUDIO (AAQAA)3

import numpy as np
import math
n_atoms = 180
n_files = 100000


path = '/home/federica_rua/Desktop/AAQAA-implicit/forces/'
directory = '/home/federica_rua/Desktop/AAQAA-implicit/forces/dataset'
save_path = '/home/federica_rua/Desktop/AAQAA-implicit/NEW/dataset_inv'

#estrazione masse dei singoli atomi della molecola
m = np.loadtxt(path+'mass.txt', dtype = 'float', skiprows = 1)
mass = np.reshape(m,[n_atoms,1]) # format masse [n_atoms,1]
forces = np.load('forces.npy')

for i in range(1,n_files+1):

    #estrazione posizioni dei singoli atomi della molecola e velocità
    data = np.loadtxt(directory+'/AAQAA-extract.rst7.%d' % i,
                      dtype = 'float', skiprows = 2 )
    data1 = np.reshape(data, [n_atoms*2,3]) #format posizioni [n_atoms*2, 3]
    vel1 = data1[n_atoms : n_atoms*2 ]
    data2 = data1[0 : n_atoms]

    #CALCOLO CENTRO DI MASSA

    cx = 0
    cy = 0
    cz = 0
    mtot = 0
    for j in range(0,n_atoms):
        mtot= mtot + mass[j]
        cx = cx + data2[j,0]*(mass[j])
        cy = cy + data2[j,1]*(mass[j])
        cz = cz + data2[j,2]*(mass[j])
```

```
c = [cx, cy, cz]/mtot
center_mass = np.reshape(c, [1,3])
data_cm = data2[:] - center_mass #traslazione nel centro di massa

#CALCOLO MOMENTO DI INERZIA

def moment_inertia_diag(data_cm,index,n_atoms):
        n_at = n_atoms
        d2 = np.power(data_cm, 2)
        i_diag = 0
        for k in range(0,n_at):
            pos = np.delete(d2[k,:],index)
            i_diag = i_diag + (mass[k])*(pos[0]+pos[1])
        return i_diag

def moment_inertia_ndiag(data_cm,index,n_atoms):
        n_at = n_atoms
        d2 = data_cm
        i_ndiag = 0
        for k in range(0,n_at):
            pos = np.delete(d2[k,:],2-index)
            i_ndiag = i_ndiag + (mass[k])*(pos[0]*pos[1])
        return i_ndiag
        # per index = 0 si calcola Ixy
        # per index = 1 si calcola Ixz
        # per index = 2 si calcola Iyz

#TENSORE DEI MOMENTI DI INERZIA
I = np.zeros((3,3))
for j in range(0,3):
    t = moment_inertia_diag(data_cm, j, n_atoms)
    I[j,j]=t

for j in range(0,3):
    y = moment_inertia_ndiag(data_cm, j, n_atoms)
    if j == 0 :
        I[0,1] = I[1,0] = -y
    elif j == 1:
        I[0,2] = I[2,0] = -y
    else:
        I[1,2] = I[2,1] = -y
```

```
    #print(I)

    #AUTOVALORI E AUTOVETTORI DEL TENSORE, ASSI PRINCIPALI
    z,f = np.linalg.eig(I)
    #print(z)
    # f autovettori

#determina l'orientazione dei versori
    if (np.dot(f[:, 0], [1, 0, 0]) < 0):
       f[:, 0] = [-element for element in f[:, 0]]

    if (np.dot(f[:, 1], [1, 0, 0]) < 0):
       f[:, 1] = [-element for element in f[:, 1]]

    if (np.dot(f[:, 2], [1, 0, 0]) < 0):
       f[:, 2] = [-element for element in f[:, 2]]

#determina l'ordine dei versori
    index_max_eig = np.argmax(z)
    index_min_eig = np.argmin(z)
    index_int_eig = int(list(set([0, 1, 2])-
                    set([index_max_eig, index_min_eig]))[0])

    #utilizzo i versori principali ottenuti
    #come colonne della matrice di trasformazione
    change_basis = np.zeros((3,3))

    change_basis[0,:] = f[:,index_max_eig]
    change_basis[1,:] = f[:,index_int_eig]
    change_basis[2,:] = f[:,index_min_eig]

    #CAMBIAMENTO DI BASE
    #(da quella del centro di massa a quello degli assi principali)

    data_inv = np.zeros((n_atoms,3))
    for j in range(0,n_atoms):
        data_inv[j] = np.reshape(np.matmul(change_basis,
                      np.reshape(data_cm[j,:],[3,1])), [1,3])
```

```
p = np.save(save_path + '/pos%d' % i, data_inv)


vel_inv = np.zeros((n_atoms,3))
for j in range(0,n_atoms):
    vel_inv[j] = np.reshape(np.matmul(change_basis,
                 np.reshape(vel1[j,:],[3,1])), [1,3])

p = np.save(save_path + '/vel%d' % i, vel_inv)

forces_inv = np.zeros((n_atoms,3))

for j in range(0,n_atoms):
    fv = forces[i-1,j]
    forces_inv[j] = np.reshape(np.matmul(change_basis,
                    np.reshape(fv,[3,1])), [1,3])

f = np.save(save_path + '/forces_inv%i' %i, forces_inv)
```

Appendice B

```
## RETE NEURALE CON OUTUPUT FORZA
## ESEMPIO (AAQAA)3 CON ARCHITTETURA 4, LR = 10*-5
import numpy as np
import matplotlib.pyplot as plt
from keras import backend as k
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
import time
import matplotlib.pyplot as plt
import numpy as np
import shutil, os
import keras
from keras.callbacks import ModelCheckpoint
from keras.callbacks import LearningRateScheduler, EarlyStopping
from keras.layers import Dense
import sys
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import callbacks
from IPython.display import clear_output
from keras.models import *
from keras.layers import Input, Conv3D, UpSampling3D, BatchNormalization, \
                         Activation, Add, concatenate, Multiply, Flatten, \
                         Dense, Dropout, MaxPooling3D, Lambda
from sklearn.preprocessing import MinMaxScaler, StandardScaler

## SETUP RETE NEURALE
n_atoms     = 180 # numero di atomi
dataset     = 100000 # dimensione dataset

delta_t = 1*10**-1 # time step
directory = '/home/ferua/NEW/AAQAA-IMP'

coords =np.load(directory+'/AAQAA-coords.npy')[:(dataset)]
# forze istantanee
forces_inv = np.load(directory+'/AAQAA-forces.npy')[:(dataset)]
# forze medie
#forces_mean = np.load(directory+'/AAQAA-force-mean.npy')[:(dataset-1)]
print(np.shape(coords))
```

```
print(np.shape(forces_inv))

m = np.loadtxt(directory+'/mass.txt', dtype = 'float', skiprows = 1)
mass = np.reshape(m, [n_atoms,1])

## ARCHITTERURA
 # Input layer per le coordinate
inputs_1    = Input((coords.shape[1],coords.shape[2]))

# si vuole appiattire gli input da (180,3) a (540)
flatten_input = Flatten()(inputs_1)
initializer = 'normal'
#si può cambiare l'architettura a piacere
dense_1=Dense(1000, kernel_initializer=initializer, activation='relu')
            (flatten_input)
dense_2=Dense(500, kernel_initializer=initializer, activation="relu")
            (dense_1)
dense_3=Dense(250, kernel_initializer=initializer, activation="relu")
            (dense_2)
dense_4=Dense(125, kernel_initializer=initializer, activation="relu")
            (dense_3)
dense_5=Dense(1, kernel_initializer=initializer, activation="relu")
            (dense_4)


force = Lambda(lambda x: k.squeeze(k.gradients(x[0], x[1]),axis=0))
             ([dense_5, inputs_1])
# calcolo del gradiente, output può essere interpretato come forza
########

model=Model(inputs=[inputs_1], outputs=force)

#######
##FUNZIONE LOSS
def root_mean_squared_error(y_true, y_pred):
  return k.sqrt(k.mean(k.square(y_pred - y_true)))

########
## COMPILAZIONE DEL MODELLO # learning rate variabile
model.compile(optimizer=Adam(learning_rate=0.00001),
             loss= root_mean_squared_error)
```

```
checkpoint = ModelCheckpoint("best_model_K.hdf5", monitor='val_loss',
                             verbose=1, save_best_only=True,
                             mode='auto', period=1)
es = EarlyStopping(monitor='val_loss', verbose=1, patience=500)

model.summary()

## TRAINING DEL MODELLO
history=model.fit([coords], forces_inv, # o forces_mean
          batch_size=500,
          epochs=1000000,
          verbose=2,
          validation_split=0.1,
          shuffle=True,
          callbacks=[checkpoint, es])

## CURVE DI APPRENDIMENTO
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.savefig('loss_forces.png')
```

Appendice C

```
## RETE NEURALE CON OUTUPUT COORDINATE
## ESEMPIO (AAQAA)3 CON ARCHITTETURA 4, LR = 10*-5
import numpy as np
import matplotlib.pyplot as plt
from keras import backend as k
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
import time
import matplotlib.pyplot as plt
import numpy as np
import shutil, os
import keras
from keras.callbacks import ModelCheckpoint
from keras.callbacks import LearningRateScheduler, EarlyStopping
from keras.layers import Dense
import sys
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import callbacks
from IPython.display import clear_output
from keras.models import *
from keras.layers import Input, Conv3D, UpSampling3D, BatchNormalization, \
                         Activation, Add, concatenate, Multiply, Flatten, \
                         Dense, Dropout, MaxPooling3D, Lambda
from sklearn.preprocessing import MinMaxScaler, StandardScaler

## SETUP RETE NEURALE
n_atoms     = 180 # numero di atomi
dataset     = 100000 # dimensione dataset

delta_t = 1*10**-1 # time step
directory = '/home/ferua/NEW/AAQAA-IMP'

coords =np.load(directory+'/AAQAA-coords.npy')[:(dataset-1)]
vel = np.load(directory+'/AAQAA-vel.npy')[:(dataset-1)]
# coordinate al time step successivo
coords_dt = np.load(directory+'/AAQAA-coords.npy')[1:(dataset)]
print(np.shape(coords))
print(np.shape(coords_dt))
```

```
print(np.shape(vel))


m = np.loadtxt(directory+'/mass.txt', dtype = 'float', skiprows = 1)
mass = np.reshape(m, [n_atoms,1])

## ARCHITTERURA
# Input layer per le coordinate
inputs_1    = Input((coords.shape[1],coords.shape[2]))
# Input layer per le velocità
inputs_2    = Input((vel.shape[1],vel.shape[2]))

# si vuole appiattire gli input da (180,3) a (540)
flatten_input = Flatten()(inputs_1)
initializer = 'normal'
# si può cambiare l'architettura a piacere
dense_1 = Dense(1000, kernel_initializer=initializer, activation='relu')
               (flatten_input)
dense_2=Dense(500, kernel_initializer=initializer, activation="relu")
             (dense_1)
dense_3=Dense(250, kernel_initializer=initializer, activation="relu")
             (dense_2)
dense_4=Dense(125, kernel_initializer=initializer, activation="relu")
             (dense_3)
dense_5=Dense(1, kernel_initializer=initializer, activation="relu")
             (dense_4)

# calcolo del gradiente, output può essere interpretato come forza
force = Lambda(lambda x: k.squeeze(k.gradients(x[0], x[1]),axis=0))
              ([dense_5, inputs_1])
########

# calcolo dell'accelerazione, velocità e coordinate
acc = Lambda(lambda x: (x[0]/x[1]))([force,mass])
vel_layer = Lambda(lambda x: (x[1] + x[0]*x[2]))([acc, inputs_2,delta_t])
pos_dt = Lambda(lambda x: (x[1]+ ((x[2]+x[0])*x[3])/2))
               ([vel_layer, inputs_1,inputs_2,delta_t])

########
model=Model(inputs=[inputs_1], outputs=force)
```

```
#######
##FUNZIONE LOSS
def root_mean_squared_error(y_true, y_pred):
  return k.sqrt(k.mean(k.square(y_pred - y_true)))

########
## COMPILAZIONE DEL MODELLO # learning rate variabile
model.compile(optimizer=Adam(learning_rate=0.00001),
                              loss= root_mean_squared_error)
checkpoint = ModelCheckpoint("best_model_K.hdf5", monitor='val_loss',
                              verbose=1,
                              save_best_only=True, mode='auto', period=1)
es = EarlyStopping(monitor='val_loss', verbose=1, patience=500)

model.summary()

## TRAINING DEL MODELLO
history=model.fit([coords, vel], coords_dt,
          batch_size=500,
          epochs=1000000,
          verbose=2,
          validation_split=0.1,
          callbacks=[checkpoint, es],
          shuffle = True)

## CURVE DI APPRENDIMENTO
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.savefig('loss_coords.png')
```

Appendice D

```
## TEST RETE NEURALE
## MODELLO FORZE E COORDINATE
## ESEMPIO (AAQAA)3
import numpy as np
from keras import backend as k
import tensorflow as tf
tf.compat.v1.disable_eager_execution()
import time
import matplotlib.pyplot as plt
import numpy as np
import shutil, os
import keras
from keras.callbacks import ModelCheckpoint
from keras.callbacks import LearningRateScheduler, EarlyStopping
from keras.layers import Dense
import sys
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import callbacks
from IPython.display import clear_output
from keras.models import *
from keras.models import load_model
from keras.layers import Input, Conv3D, UpSampling3D, BatchNormalization, \
                         Activation, Add, concatenate, Multiply, Flatten, \
                         Dense, Dropout, MaxPooling3D, Lambda

# DATASET

n_atoms     = 180 # number of atoms
dataset     = 100000 # width of the dataset


directory = '/home/ferua/NEW/AAQAA-IMP'

coords =np.load(directory+'/AAQAA-coords.npy')[:(dataset)]
forces_inv = np.load(directory+'/AAQAA-forces.npy')[:(dataset)]

######
# Alternativa modello coordinate
```

```
#vel = np.load(directory+'/AAQAA-vel.npy')[:(dataset-1)]
#coords_dt = np.load(directory+'/AAQAA-coords.npy')[1:(dataset)]
#####
m = np.loadtxt(directory+'/mass.txt', dtype = 'float', skiprows = 1)
mass = np.reshape(m, [n_atoms,1])


## CARICO DEL BEST MODEL OTTENUTO DURANTE IL TRAINING

# Richiamo funzione loss
def root_mean_squared_error(y_true, y_pred):
  return k.sqrt(k.mean(k.square(y_pred - y_true)))

model = keras.models.load_model('best_model_K.hdf5',
       custom_objects = {'root_mean_squared_error': root_mean_squared_error })


## OTTENERE VALORI PREDITTIVI
forces_predict = model.predict([coords], steps = 1)

f = np.save('forces_predict', forces_predict)
print(np.shape(forces_predict))

######
# modello coordinate
coords_predict = model.predict([coords, vel], steps = 1)
np.save('coords_predict.npy', coords_predict)
######

## CALCOLO LOSS
los =model.evaluate([coords], forces_inv, verbose = 0)
######
# modello coordinate
los =model.evaluate([coords, vel], coords_dt, verbose = 0)
######

print(los)
```

Appendice E

```
## CREAZIONE FILE PDB
## CASO (AAQAA)3
## FRAME 95001

import numpy as np

n_atoms = 180
path = '/home/federica/Desktop/AAQAA-implicit/check/create_file_pdb/'

col3 = np.loadtxt(path+'col3.txt', dtype = 'str', unpack = True)
col2 = np.loadtxt(path+'col2.txt', dtype = 'str', unpack = True)
col4 = np.loadtxt(path+'col4.txt', dtype = 'str', unpack = True)
col5 = np.loadtxt(path+'col5.txt', dtype = 'str', unpack = True)
#col_end = np.loadtxt('col-end.txt', dtype = 'str', unpack = True)

coords = np.zeros((n_atoms,3))
coords = np.load('AAQAA-coords95001.npy')
#coords = c[0]
print(coords[0])
print('+++++++')
for i in range(0,n_atoms):
  for j in range (0,3):
    #t[i] = int(100)
    coords[i,j] = float("{:.3f}".format(coords[i,j]))


f = open("AAQAA-coords95001.pdb", "w")

for i in range (0, n_atoms):
  f.write("ATOM")
  f.write("{:>7}".format(col2[i]))
  f.write("{:^6}".format(col3[i]))
  f.write("{}".format(col4[i]))
  f.write("{:>5}".format(col5[i]))
  f.write("{:>13}{:8}{:8}".format(*coords[i]))
  f.write("{:>6}".format('1.00'))
  f.write("{:>6}".format('0.00'))
  #f.write("{:>12}".format(col_end[i]))
  f.write('\n')
```

```
f.write("TER")
f.write("{:>8}".format('181'))
f.write("{:>9}".format('NH2'))
f.write("{:>5}".format('17'))
f.write('\n')
f.write("END")

## CREAZIONE FILE PDB
## CASO DIALANINA
## FRAME 37501

import numpy as np

n_atoms = 23
path = '/home/federica/Desktop/2ala-implicit/check/create_file_pdb/'


col3 = np.loadtxt(path+'col3.txt', dtype = 'str', unpack = True)
col2 = np.loadtxt(path+'col2.txt', dtype = 'str', unpack = True)
col4 = np.loadtxt(path+'col4.txt', dtype = 'str', unpack = True)
col5 = np.loadtxt(path+'col5.txt', dtype = 'str', unpack = True)

t = np.zeros((n_atoms,1))
coords = np.load('2ala-coords37501.npy')

print('+++++++')
for i in range(0,n_atoms):
  for j in range (0,3):
    t[i] = int(100)
    coords[i,j] = float("{:.3f}".format(coords[i,j]))


f = open("2ala-coords37501.pdb", "w")

for i in range (0, n_atoms):
  f.write("ATOM")
  f.write("{:>7}".format(col2[i]))
  f.write("{:^6}".format(col3[i]))
  f.write("{}".format(col4[i]))
  f.write("{:>5}".format(col5[i]))
```

```
  f.write("{:>13}{:8}{:8}".format(*coords[i]))
  f.write("{:>6}".format('1.00'))
  f.write("{:>6}".format('0.00'))
  #f.write("{:>12}".format(col_end[i]))
  f.write('\n')

f.write("TER")
f.write("{:>8}".format('24'))
f.write("{:>9}".format('ALA'))
f.write("{:>5}".format('2'))
f.write('\n')
f.write("END")
```

Riferimenti bibliografici

[1] V. Venkatasubramanian, The promise of artificial intelligence in chemical engineering: Is it here, finally, AIChE Journal 65 (2) (2019) 466–478.

[2] P. Hamet, J. Tremblay, Artificial intelligence in medicine, Metabolism 69 (2017) S36–S40.

[3] B. Mahesh, Machine learning algorithms-a review, International Journal of Science and Research (IJSR).[Internet] 9 (2020) 381–386.

[4] A. Géron, Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems, O'Reilly Media, Inc., 2019.

[5] P. A. M. Dirac, Quantum mechanics of many-electron systems, Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character 123 (792) (1929) 714–733.

[6] F. Noé, A. Tkatchenko, K.-R. Müller, C. Clementi, Machine learning for molecular simulation, Annual review of physical chemistry 71 (2020) 361–390.

[7] M. P. Allen, et al., Introduction to molecular dynamics simulation, Computational soft matter: from synthetic polymers to proteins 23 (1) (2004) 1–28.

[8] J.-P. Ryckaert, G. Ciccotti, H. J. Berendsen, Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes, Journal of computational physics 23 (3) (1977) 327–341.

[9] H. C. Andersen, Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations, Journal of computational Physics 52 (1) (1983) 24–34.

[10] Y. Chen, A. Krämer, N. E. Charron, B. E. Husic, C. Clementi, F. Noé, Machine learning implicit solvation for molecular dynamics, The Journal of Chemical Physics 155 (8) (2021) 084101.

[11] R. Anandakrishnan, A. Drozdetski, R. C. Walker, A. V. Onufriev, Speed of conformational change: comparing explicit and implicit solvent molecular dynamics simulations, Biophysical journal 108 (5) (2015) 1153–1164.

[12] J. Behler, M. Parrinello, Generalized neural-network representation of high-dimensional potential-energy surfaces, Physical review letters 98 (14) (2007) 146401.

[13] S. Chmiela, H. E. Sauceda, K.-R. Müller, A. Tkatchenko, Towards exact molecular dynamics simulations with machine-learned force fields, Nature communications 9 (1) (2018) 1–10.

[14] J. Wang, S. Olsson, C. Wehmeyer, A. Pérez, N. E. Charron, G. De Fabritiis, F. Noé, C. Clementi, Machine learning of coarse-grained molecular dynamics force fields, ACS central science 5 (5) (2019) 755–767.

[15] E. Schneider, L. Dai, R. Q. Topper, C. Drechsel-Grau, M. E. Tuckerman, Stochastic neural network approach for learning high-dimensional free energy surfaces, Physical review letters 119 (15) (2017) 150601.

[16] A. Mardt, L. Pasquali, H. Wu, F. Noé, Vampnets for deep learning of molecular kinetics, Nature communications 9 (1) (2018) 1–11.

[17] D. A. Pearlman, D. A. Case, J. W. Caldwell, W. S. Ross, T. E. Cheatham III, S. DeBolt, D. Ferguson, G. Seibel, P. Kollman, Amber, a package of computer programs for applying molecular mechanics, normal mode analysis, molecular dynamics and free energy calculations to simulate the structural and energetic properties of molecules, Computer Physics Communications 91 (1-3) (1995) 1–41.

[18] P. Robustelli, S. Piana, D. E. Shaw, Developing a molecular dynamics force field for both folded and disordered protein states, Proceedings of the National Academy of Sciences 115 (21) (2018) E4758–E4766.

[19] A. Onufriev, D. Bashford, D. A. Case, Exploring protein native states and large-scale conformational changes with a modified generalized born model, Proteins: Structure, Function, and Bioinformatics 55 (2) (2004) 383–394.

[20] G. A. Tribello, M. Bonomi, D. Branduardi, C. Camilloni, G. Bussi, Plumed 2: New feathers for an old bird, Computer Physics Communications 185 (2) (2014) 604–613.

[21] F. Pietrucci, A. Laio, A collective variable for the efficient exploration of protein beta-sheet structures: application to sh3 and gb1, Journal of Chemical Theory and Computation 5 (9) (2009) 2197–2201.

[22] A. Arsiccio, J.-E. Shea, Protein cold denaturation in implicit solvent simulations: A transfer free energy approach, The Journal of Physical Chemistry B 125 (20) (2021) 5222–5232.

[23] G. Ramachandran, C. Ramakrishnan, V. Sasisekharan, Stereochemistry of polypeptide chain configurations, Journal of Molecular Biology 7 (1963) 95–99.

[24] R. A. Laskowski, N. Furnham, J. M. Thornton, The ramachandran plot and protein structure validation, in: Biomolecular Forms and Functions: A Celebration of 50 Years of the Ramachandran Map, World Scientific, 2013, pp. 62–75.

[25] J. S. Richardson, The anatomy and taxonomy of protein structure, Advances in protein chemistry 34 (1981) 167–339.

[26] Grafico di ramachandran, `https://it.wikipedia.org/wiki/Grafico di Ramachandran`, accesso: 2022-02-07.

[27] G. Piatetsky-Shapiro, Knowledge discovery in real databases: A report on the ijcai-89 workshop, AI magazine 11 (4) (1990) 68–68.

[28] R. S. Michalski, Understanding the nature of learning: Issues and research directions, Morgan Kaufmann Publishers, 1986.

[29] Y. Kosmann-Schwarzbach, The noether theorems, in: The Noether Theorems, Springer, 2011, pp. 55–64.

[30] J. Ling, R. Jones, J. Templeton, Machine learning strategies for systems with invariance properties, Journal of Computational Physics 318 (2016) 22–35.

[31] Lambda layer, `https://keras.io/api/layers/core_layers/lambda/`, accesso: 2022-02-07.

[32] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980.

[33] J. Gareth, W. Daniela, H. Trevor, T. Robert, An introduction to statistical learning: with applications in R, Spinger, 2013.

[34] Callbacks, `https://keras.io/api/callbacks/`, accesso: 2022-02-07.

[35] P. R. Norvig, Artificial intelligence: a modern approach, Prentice Hall Upper Saddle River, NJ, USA:, 2002.

[36] How to use learning curves to diagnose machine learning model performance, https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/, accesso: 2022-02-10.

[37] I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.

[38] B. E. Husic, N. E. Charron, D. Lemm, J. Wang, A. Pérez, M. Majewski, A. Krämer, Y. Chen, S. Olsson, G. de Fabritiis, F. Noeé, C. Clementi, Coarse graining molecular dynamics with graph neural networks, The Journal of Chemical Physics 153 (19) (2020) 194101.

Ringraziamenti

In conclusione di questo lavoro di tesi vorrei cogliere l'occasione per ringraziare chi ha reso possibile tutto ciò.

Ringrazio il Prof. Roberto Pisano per avermi dato l'opportunità di lavorare a questo progetto di tesi, per avermi quindi concesso di scoprire e di innamorarmi di un mondo ancora inesplorato nel mio percorso di studi. Ringrazio il Prof. Gianluca Boccardo per avermi seguita passo passo nello sviluppo di questo lavoro e di avermi guidata nel mio viaggio alla scoperta del Machine Learning. Ringrazio il dott. Andrea Arsiccio, o meglio Andrea, per essere stato una fonte di infinita conoscenza e di pazienza, anche forse troppa nei miei confronti, e di avermi illuminato sul campo della dinamica molecolare. Ringrazio Agnese Marcato per avermi seguita e aiutata a compiere i miei primi passi nel mondo delle Reti Neurali e che neanche un oceano le ha impedito di continuare ad aiutarmi.

Vorrei ringraziare anche chi ha reso veramente possibile tutto quanto: la mia famiglia. Un grazie dal profondo del cuore ai miei genitori e a mio fratello Dario, che mi hanno sostenuta sempre e comunque. Hanno gioito insieme a me per i miei successi accademici e non mi hanno abbandonato quando ho affrontato momenti difficili, le mie ancore di salvezza.

Un grazie è doveroso alle mie migliori amiche Sara e Sara, meglio note come Cava e Situ, che mi hanno accompagnato in questo percorso. La convivenza a Torino di tre aspiranti ingegneri è stata una delle avventure più belle che io abbia mai sperimentato. Siete state un regalo prezioso e non potrò mai ringraziarvi abbastanza per la vostra amicizia.

Infine, voglio dedicare questa mia tesi a mio nonno Bernardo, o semplicemente Paiei, che purtroppo non è più qui con noi per essere testimone di questo mio traguardo ma che è uno dei motivi per cui sono arrivata alla fine di questo mio percorso.