UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Fisica

Studio degli eventi che inducono il folding di proteine con metodi di Machine Learning

Relatore: **Prof. Guido Tiana**

Tesi di Laurea Triennale di:
Andrea Scaioli
Matricola: 979208

Anno Accademico 2022/2023

Alla mia famiglia
Marco, Daniela
Maria
Lorenzo

Due cose nella mia vita sono importanti. La prima è questa: che proprio per quel che vi ho detto, il gusto della vita non è negato a chi sbaglia, ma a chi non ha un senso dell'infinito, del destino, dell'ideale, del Mistero presente, perché allora il problema non è sbagliare o non sbagliare. Il gusto della vita non è negato a chi sbaglia: è negato a chi non ha un nesso con il Destino che fa le cose, con il Mistero presente. Per cui tutto è un'ipotesi positiva, il tempo che per tutti è sinonimo di decadenza, lavora in positivo. Se guardo la mia vita, che razza di roba è successa! Dico sempre: se è successo così fino adesso, immaginiamoci cosa succederà nel futuro! Ne vedremo delle belle. È interessante, no? È un'avventura

E. Piccinini

Indice

Introduzione

In questo progetto di tesi si vuole studiare tramite simulazioni di dinamica molecolare e metodi di Machine Learning il folding delle proteine, in particolare ho sviluppato un metodo per individuare quali sono i legami più significativi tra amminoacidi per il processo di ripiegamento della proteina G (1PGB) e della emoagglutinina (2HMG).
Le proteine sono catene di amminoacidi uniti tra loro da legami peptidici che, in determinate condizioni biologiche, passano da uno stato inizialmente dispiegato e privo di interazioni a una struttura proteica tridimensionale [1] in cui possono svolgere le loro funzioni. Tale processo prende il nome di folding proteico ed è governato da contatti tra amminoacidi non consecutivi.
Sperimentalmente si è osservato che sono pochi i contatti che caratterizzano il processo di folding, infatti proteine con più del 35% della catena uguale hanno la stessa struttura tridimensionale [2].

Il folding è un processo stocastico complesso che è significativo comprendere in diversi campi, come la biologia molecolare, la medicina e la progettazione di farmaci. Questo processo coinvolge una grande quantità di interazioni interatomiche, infatti gli atomi in una proteina, che interagiscono anche con l'ambiente circostante, possono essere diverse migliaia.

Poiché è difficile studiare sperimentalmente ed in dettaglio il processo sono ampiamente utilizzati metodi di simulazione computazionale, ma i parametri di cui tener conto sono innumerevoli e non sempre è possibile effettuare simulazioni in tempi ridotti. Da anni sono stati impiegati nello studio del ripiegamento delle proteine numerosi metodi di Machine Learning [3] che hanno permesso di predirne la struttura tridimensionale, la dinamica e, di conseguenza, le funzioni biologiche svolte.

Questo progetto si inserisce in questo contesto di sviluppo. Sono state eseguite simulazioni di traiettorie di folding delle proteine 1PGB e 2HMG. Dalle traiettorie sono state ottenute mappe di contatto a diversi momenti del processo. In seguito è stata progettata ed allenata una rete neurale feed forward che riceve come input una mappa dei contatti e predice il tempo normalizzato del processo di folding a cui è stata estratta la mappa. Dal modello ottenuto, è stato possibile determinare i contatti fondamentali durante il processo di folding tramite un algoritmo di ottimizzazione stocastica. Fornendo un tempo normalizzato come

input, tle algoritmo restituisce la più probabile mappa di contatto come output in funzione di determinate condizioni iniziali. Verificata l'efficacia del modello di Machine Learning anche impiegando un training dataset composto da una singola traiettoria di folding, si è ripetuto il processo con la emoagglutinina: una proteina formata da 6 monomeri e quindi più complessa da modellizzare con successo.

Anche per l'emoagglutinina è stata simulata una traiettoria di ripiegamento, è stato creato un modello di Machine Learning e sono state realizzate le mappe relative a tempi differenti con un algoritmo di ottimizzazione.

Capitolo 1

Proteine e processo di folding

1.1 Proteina: struttura, funzioni e ipotesi termodinamica

Le proteine [4] sono macromolecole biologiche che svolgono funzioni essenziali per la vita di un organismo. Sono costituite da catene di amminoacidi legati tra loro da legami peptidici. La sequenza di amminoacidi di una proteina è determinata dal DNA che codifica la sua struttura e funzione.
Esistono 20 tipi diversi di amminoacidi che compongono la catena polipeptidica, ognuno di questi è formato da una parte fissa, detta spina dorsale (*Backbone*), composta da un gruppo amminico e un gruppo carbossilico, e una catena laterale variabile (*Side chain*). Quest'ultima è ciò che caratterizza l'amminoacido e viene chiamata anche *Residuo*. In Fig. 1.1 è mostrata la struttura chimica di un amminoacido.

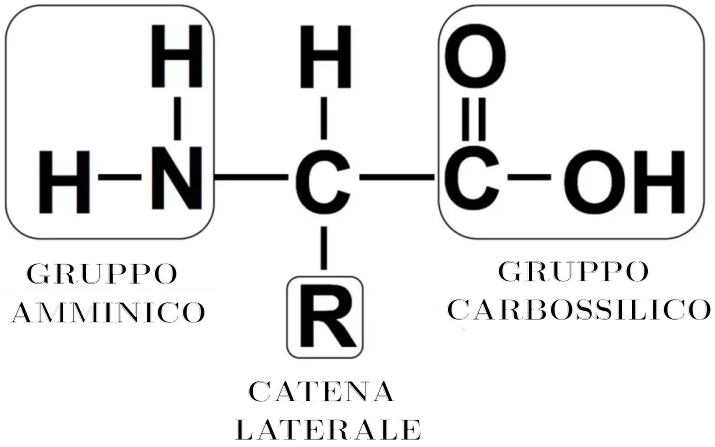


Figura 1.1: Il gruppo amminico, il carbonio e l'idrogeno centrali e il gruppo carbossilico formano la spina dorsale di ogni amminoacido. La parte variabile (Residuo) è rappresentato dalla *R*.

Le proteine svolgono un'ampia gamma di funzioni [5] che si possono suddividere in 4 macro-categorie:

1. **Strutturali**: forniscono struttura e sostegno alle cellule e ai tessuti. Ad esempio, la cheratina, che si trova nei capelli, nelle unghie e nella pelle, fornisce resistenza ed elasticità ai tessuti.

2. **Regolazione**: regolano le reazioni chimiche nel corpo. Gli enzimi, che catalizzano le reazioni chimiche, sono un esempio di proteine regolatrici.

3. **Trasporto**: sono le proteine che trasportano sostanze negli organismi. L'emoglobina, presente nei globuli rossi, trasporta ossigeno in tutto l'organismo.

4. **Difesa**: un esempio di proteine di difesa sono gli anticorpi: proteine che combattono virus e batteri.

La struttura della proteina è una proprietà strettamente legata alle funzioni che svolge. La forma e la disposizione degli amminoacidi in una proteina determinano la sua capacità di legarsi ad altre molecole, catalizzare reazioni chimiche e svolgere altre funzioni biologiche.
La struttura di una proteina si studia su tre livelli differenti: struttura primaria, secondaria e terziaria, illustrate in Fig. 1.2.

La struttura primaria è la sequenza di amminoacidi che compongono la proteina ed è il primo stadio descrittivo della struttura di quest'ultima. La struttura secondaria descrive le sotto-strutture regolari presenti nella proteina come β-$forcine$ o α-$eliche$ che si formano tramite legami a idrogeno, nei quali un atomo di idrogeno recante una parziale carica elettrica positiva si lega con un doppietto elettronico solitario di un elemento fortemente elettronegativo. Mentre un'α-$elica$ è caratterizzata da una spirale destrorsa formata da una catena polipeptidica piegata in una struttura ripetitiva, una β-$forcina$ è caratterizzata da una disposizione planare di segmenti polipeptidici che si piegano ripetutamente formando una struttura a foglietto. In un'α-$elica$, i legami idrogeno si formano tra il gruppo amminico e il gruppo carbossilico situato quattro posizioni più avanti nella sequenza di amminoacidi, mentre in una β-$forcina$ i segmenti polipeptidici in un foglio beta sono collegati da legami idrogeno tra i gruppi amminici e carbossilici di amminoacidi differenti. Queste strutture conferiscono stabilità alla proteina e possono essere trovate in molte proteine. Infine, la struttura terziaria è la struttura globale tridimensionale della proteina.

L'ipotesi su cui si fonda lo studio e la trattazione di questi sistemi biologici complessi è l' ipotesi termodinamica di Anfinsen. Secondo l'ipotesi termodinamica una proteina ha una struttura tridimensionale sola unica che corrisponde al minimo di energia libera. Questo stato di equilibrio, unico e cineticamente accessibile, viene chiamato *stato nativo*. Nel 1937 Anfinsen studia la nucleasi stafilococcica [7][8][9] e nota che in condizioni biologiche è attiva, mentre denaturando le proteine con urea (una sostanza chimica che può rompere legami non covalenti, come

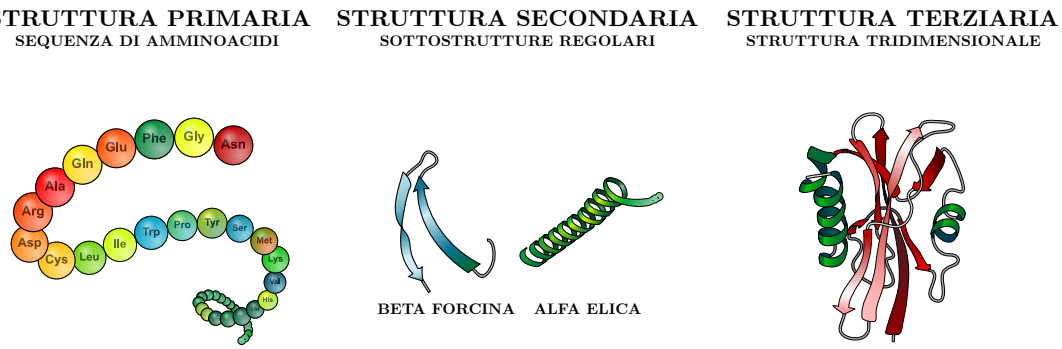


Figura 1.2: Le tre strutture primaria, secondaria e terziaria di una proteina. L'immagine è tratta da [6].

quelli idrogeno, presenti nelle proteine) le interazioni tra gli amminoacidi si indeboliscono [10] e lo stato di equilibrio si sposta verso altre conformazioni dove la proteina non svolge più le sue funzioni biologiche, questo stato prende il nome di *stato denaturato*. Riportando il sistema in condizioni biologiche la proteina ritorna ad esercitare le sue attività [11].
Durante l'esperimento, Anfinsen denatura e rinatura la proteina così che oscilli tra due stati differenti [12]. Così facendo, scopre che la struttura primaria delle proteine, cioè la sequenza di amminoacidi, è sufficiente a determinare completamente la struttura tridimensionale per proteine in grado di transire dallo stato nativo allo stato denaturato in vitro.

Mentre lo stato denaturato è uno stato ad alta entropia, poiché è un insieme di molte conformazioni disordinate, una proteina isolata in un ambiente a temperatura e volume costante si ripiega sempre nel suo unico stato nativo. Per questo motivo, tale stato viene identificato come un minimo globale ed isolato dell'energia libera di Helmholtz:

$$F = U - TS \tag{1.1}$$

dove U è l'energia interna del sistema, T la temperatura e S l'entropia.
Lo stato nativo della proteina è caratterizzato da entropia nulla, infatti la sua conformazione è unica, cioè esiste un unico microstato [13]:

$$S = K_B log(\Omega) = 0 \tag{1.2}$$

dove K_B è la costante di Boltzmann e Ω il numero di microstati relativi ad un macrostato.

É complesso studiare come l'informazione presente nella catena di amminoacidi permetta di determinare la geometria della proteina perché il processo di folding, durante il quale si passa da stato denaturato a stato nativo, si sviluppa simultaneamente in diverse aree della catena ed è quindi un processo collettivo [14].
Sebbene una sequenza determini univocamente la geometria della proteina, il problema inverso non è univoco, infatti, una struttura tridimensionale può essere

ottenuta da un enorme numero di sequenze differenti.
Sperimentalmente, si è osservato che proteine con più del 35% della sequenza uguale formano la stessa struttura tridimensionale. Le proteine sono quindi molto tolleranti alle mutazioni puntiformi poiché in un gran numero di siti le variazioni non hanno effetto sul processo di ripiegamento.
La struttura secondaria della proteina non dipende semplicemente dal tipo di amminoacido che la compone, ma dalla globalità del sistema. Per questo il processo di folding si dice un processo cooperativo, infatti gli stessi amminoacidi in diverse proteine possono formare β-$forcine$ o α-$eliche$.

Al termine del processo di folding una proteina ha formato la struttura tridimensionale che le permette di svolgere importanti funzioni biologiche specifiche. Solo le proteine correttamente ripiegate hanno una stabilità a lungo termine in ambienti biologici affollati e sono in grado di interagire selettivamente con i loro partner naturali. Non sorprende quindi che il mancato ripiegamento corretto delle proteine, o il loro mancato mantenimento, sia all'origine di un'ampia varietà di condizioni patologiche.

Studiare il processo di folding delle proteine è molto importante per diversi motivi, in particolare, la struttura tridimensionale di una proteina è strettamente correlata alla sua funzione biologica. I legami tra amminoacidi determinano la forma tridimensionale delle proteine, che a sua volta influisce sulla capacità di interagire con altre molecole e svolgere specifiche funzioni nel corpo.
Inoltre, esistono molti disturbi e malattie umane legati a mutazioni che influenzano il folding delle proteine. Queste mutazioni possono compromettere la struttura tridimensionale corretta delle proteine, portando a disfunzioni biochimiche e patologie. Ad esempio, malattie come Alzheimer, Parkinson e fibrosi cistica sono associate a difetti nel folding proteico [15].
Grazie alla conoscenza del processo di folding è possibile progettare e modificare proteine per scopi specifici, come la produzione di farmaci, l'industria alimentare, la biotecnologia e altro ancora. Manipolare i legami tra amminoacidi può consentire la creazione di proteine con nuove funzioni o proprietà desiderate [16].

In sintesi, il folding è un processo cooperativo nel quale la proteina passa dallo stato denaturato allo stato nativo. Lo stato denaturato è caratterizzato da alta entropia e si raggiunge ad alte temperature, mentre lo stato nativo ha entropia nulla e si manifesta a basse temperature. Allo stato denaturato, il sistema perde la capacità di eseguire le sue funzioni biologiche e assume una struttura disordinata e caotica che occupa uno spazio maggiore rispetto a quando si trova allo stato nativo, in cui invece svolge normalmente le sue attività biologiche.

1.2 Transizioni di fase, temperatura di folding e RMSD

Sperimentalmente emerge che le proteine mostrano una transizione di fase del I° ordine allo stato denaturato, identificabile con un picco nel calore specifico mediante esperimenti di calorimetria con T compresa tra i 40°C e i 70°C.
Si hanno transizioni di fase del I° ordine quando si ha una discontinuità nella derivata I° di Z, cioè nella energia media $\langle E\rangle = \frac{\partial}{\partial\beta} log(Z)$. Si ha una transizione di fase del II° ordine se è presente una discontinuità nella derivata seconda di Z, cioè nel calore specifico $C_v(T) = \frac{\partial}{\partial T}\langle E\rangle = \frac{\partial^2}{\partial\beta^2} log(Z)$.

Nel limite termodinamico la transizione di fase è a gradino, mentre per sistemi finiti si hanno andamenti più graduali, come mostrato in Fig. 1.3.
Studiando il grafico di $S(E)$ in Fig. 1.3 per una proteina si può osservare una regione convessa che non corrisponde a stati di equilibrio e nella quale sono presenti meno stati di quelli che ci si aspetta, questo è dovuto alla *cooperatività* del sistema. La regione convessa presenta un basso numero di stati che non possono essere all'equilibrio, infatti:

$$\frac{\partial^2 S}{\partial E^2} = \frac{\partial}{\partial E}\frac{1}{T} = \frac{\partial T}{\partial E}\frac{1}{dT}\frac{1}{T} = -\frac{1}{T^2 C_v} < 0$$

Per trovare l'energia media ad una data temperatura basta sfruttare $\frac{\partial S}{\partial E} = \frac{1}{T}$. Il cambio di concavità di $S(E)$ fa sì che esista una retta a cui corrispondono due energie differenti E_1 ed E_2. Questa temperatura viene e chiamata *temperatura di folding* T_F.
Le proteine piccole mostrano un transizione di fase singola tra stato nativo e stato denaturato alla temperatura di folding T_f perché, abbassando la temperatura, il sistema tende a stabilizzarsi grazie a diverse parti che cooperano tra loro per mantenere una configurazione stabile.

L'energia libera di Helmholtz $F(E) = E - TS(E)$ presenta dei minimi in corrispondenza degli stati con massima probabilità di equilibrio come mostrato in in figura 1.4. Il primo minimo (N) è lo stato nativo ($S = 0$ ed E bassa), mentre il secondo minimo (D) è lo stato denaturato dovuto al bilanciamento tra i grandi valori di S ed E. Tra i due minimi è presente un massimo a causa della cooperatività del sistema. Il minimo assoluto di $F(E)$ dipende dalla temperatura, a T basse il termine $-TS$ è trascurabile e prevale lo stato nativo, ad alte temperature prevale il termine entropico e quindi lo stato denaturato, mentre quando $T = T_f$ i due stati sono equiprobabili.

Una grandezza importante per valutare cambiamenti strutturali è l'*RMSD* acronimo di *Root Mean Square Deviation*:

$$RMDS = \sqrt{\frac{1}{M}\sum_{i=1}^{N} m_i ||\vec{r}_i(t_1) - \vec{r}_i(t_2)||^2} \tag{1.3}$$

dove $M = \sum_{i=1}^{N} m_i$ è la massa totale della catena, N il numero di atomi, $\vec{r}_i(t)$ la posizione dell' atomo i al tempo t.
L'RMSD è una misura statistica che indica la deviazione media delle posizioni degli atomi o dei residui da una struttura di riferimento. Nella bioinformatica e nella biologia computazionale, l'RMSD è un parametro fondamentale per valutare la qualità delle predizioni di strutture proteiche.
Calcolando l'RMSD in diverse condizioni sperimentali (ad esempio, a diverse temperature o concentrazioni di denaturante), è possibile confrontare la stabilità relativa della proteina in diverse situazioni. Questo può fornire informazioni sulla dipendenza della stabilità strutturale dalla temperatura, dal pH, dai solventi o da altre variabili.
Studiando l'RMSD in una traiettoria dove la proteina passa da stato nativo a denaturato si ha che l'RMSD aumenta al dispiegarsi della proteina e sarà massimo al raggiungimento dello stato denaturato.

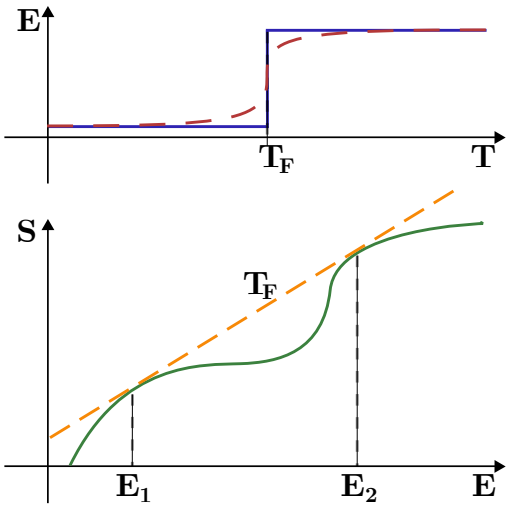


Figura 1.3: Il grafico in alto mostra la transizione di fase da stato nativo a stato denaturato in un grafico $E(T)$. Alla temperatura di folding lo stato passa da uno stato ad alta energia ad uno stato a bassa energia.
Il grafico in basso mostra la transizione di fase in un grafico $S(E)$. la transizione avviene da uno stato ad alta entropia ad uno stato a bassa entropia.

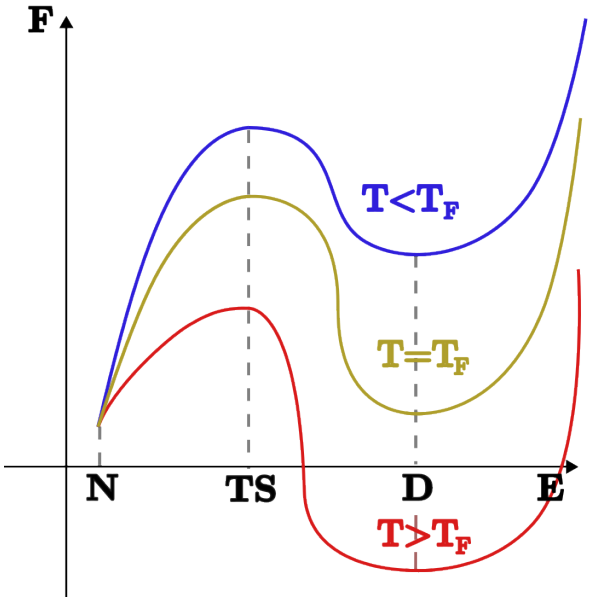


Figura 1.4: All'aumentare dell'energia del sistema lo stato di equilibrio si sposta dallo stato nativo N allo stato denaturato D. Tra i due stati è presente un massimo locale causato dalla cooperatività del sistema, infatti gli stati intermedi non sono stati di equilibrio.
Al variare della temperatura cambia il minimo globale del sistema: per temperature inferiori alla temperatura di folding il minimo globale di F è allo stato nativo, mentre per temperature superiori alla temperatura di folding il minimo globale di F è allo stato denaturato.

1.3 Interazioni tra amminoacidi

La struttura e la dinamica delle proteine è determinata dalle interazioni presenti nel sistema. Le interazioni tra gli amminoacidi sono di diversi tipi, le due più rilevanti sono:

- **Interazione Elettrostatica**: si trova tra alcuni amminoacidi che presentano una carica netta dipendente dal pH della soluzione in cui si trovano.
  Questi amminoacidi si legano con interazione coulombiana $U = \frac{q_1 q_2}{4\pi\varepsilon_0\varepsilon_r d}$ dove q_1 e q_2 sono le cariche, ε_0 e ε_r le costanti dielettriche nel vuoto e nel materiale e d la distanza tra gli amminoacidi.
  In presenza di un solvente o in un bagno termico gli ioni delle proteine vengono circondati da ioni di cariche opposte, così si viene a formare un multipolo. Imponendo che il potenziale coulombiano soddisfi l'equazione di Poisson e che la distribuzione di carica all'equilibrio sia di Boltzmann si ottiene l'equazione di Poisson-Boltzmann.

$$\nabla^2\phi(x) = -\frac{1}{\varepsilon}\sum_{i=i}^{N} q_i\rho_{i0}e^{-\beta q_i\phi(x)} \tag{1.4}$$

  Dove $\phi(x)$ è il potenziale elettrostatico, ε la costante dielettrica del solvente e $q_i\rho_i(x)$ la distribuzione di carica dell'atomo i.

Ad alte temperature, se la carica totale è nulla si ha che $\phi(x) = \frac{1}{r}e^{-\frac{r}{\lambda_D}}$ dove $\lambda_D = (\frac{\varepsilon T}{\sum_i q_i^2 \rho_{i0}})^{\frac{1}{2}}$ è la lunghezza di Debye. Quindi il potenziale elettrostatico decresce esponenzialmente in funzione della distanza e viene completamente schermato ad una distanza pari alla lunghezza di Debye.

- **Interazioni di Van der Waals**: sono interazioni quantistiche associate a fluttuazioni di punto zero delle molecole.
  In un sistema hamiltoniano di due molecole nel loro stato fondamentale che non hanno momento di dipolo, la carica può oscillare per produrre un dipolo lungo l'asse z. Consideriamo quindi le fluttuazioni di dipolo delle due molecole e l'interazione dipolo-dipolo in una hamiltoniana:

$$H = \frac{1}{2\alpha}z_1^2 + \frac{1}{2\alpha}z_2^2 + \frac{1}{R^3}z_1 z_2$$

  dove α è la polarizzazione delle molecole ed R la distanza tra i due dipoli. Passando in coordinate radiali $z_+ = \frac{1}{2}(z_1 + z_2)$ e $z_- = \frac{1}{2}(z_1 - z_2)$ l'hamiltoniana assume la forma:

$$U = \frac{1}{\alpha}\left(1 + \frac{\alpha}{R^3}\right)z_+^2 + \frac{1}{\alpha}\left(1 - \frac{\alpha}{R^3}\right)z_-^2$$

  L'energia allo stato fondamentale di questa coppia di oscillatori armonici quantistici è $E = \frac{1}{2}(V_{0+} + V_{0-})$:

$$V_{0\pm} = \left[\frac{1}{m\alpha}(1 \pm \frac{\alpha}{R^3})\right]^{\frac{1}{2}} \approx (m\alpha)^{\frac{1}{2}}\left[1 \pm \frac{\alpha}{R^3} - \frac{\alpha^2}{2R^6}\right]$$

  nel caso in cui valga che $R \gg \alpha^{\frac{1}{3}}$, caso fisicamente verificato.
  Il potenziale assume la forma $E = \hbar(m\alpha)^{\frac{1}{2}}(1 - \frac{\alpha^2}{R^6}) + \frac{1}{R^{12}}$, noto come potenziale di Lennard-Jones, che può essere attrattivo o repulsivo a seconda della distanza e i cui parametri numerici dipendono dai tipi di amminoacidi considerati.
  Approssimando il potenziale di Lennard-Jones ad una buca di pontenziale anche in questo caso è possibile, come per il multipolo elettrico, definire una distanza di troncamento (cut-off).

1.4 Dinamica ed equazioni di Langevin

Per descrivere la dinamica di una proteina le equazioni di Newton non sono adatte, perché bisognerebbe scrivere tali equazioni per ogni atomo. Dato un sistema di N atomi, si hanno $6N$ equazioni, la maggior parte delle quali sono associate ad atomi di solvente, i cui movimenti non sono interessanti per la dinamica della proteina. É preferibile rinunciare ad una descrizione molecolare del solvente e fornire una descrizione efficace del suo effetto sulla proteina, come la modifica delle proprietà dielettriche del mezzo, l'interazione idrofobica, l'attrito e lo scambio di energia.

In questo modo si descrive la dinamica della proteina come un processo stocastico nel quale si passa tramite stati metastabili.
Si introducono quindi le equazioni di Langevin:

$$\dot{\mathbf{p}}_\mathbf{i} = \mathbf{f}_\mathbf{i} - \frac{\gamma}{m_i}\mathbf{p}_\mathbf{i} + \eta(t) \tag{1.5}$$

dove $\dot{\mathbf{p}}_\mathbf{i}$ è la derivatqa del momento rispetto al tempo, $\gamma = 6\pi\eta_S R$ è il coefficiente di attrito viscoso che smorza l'accelerazione atomica calcolato tramite la legge di Stokes dove η_S è il è la viscosità del solvente e R il raggio dell'amminoacido, f_i sono le forze tra gli amminoacidi descritte precedentemente ed $\eta(t)$ è una variabile stocastica con una certa distribuzione che descrive gli urti tra solvente e proteina.

Ipotizzando che il sistema sia isotropo, che il numero di atomi sia elevato e che gli urti tra le molecole siano totalmente scorrelati si può applicare il teorema del limite centrale alla forza $\eta(t)$ su scale di tempo dell'ordine di quelle del moto della proteina per cui $\eta(t)$ ha una distribuzione gaussiana.
Gli urti tra solvente e amminoacidi possono essere trattati in maniera deterministica, ma descrivere il moto di tante particelle è complesso, per questo si possono supporre gli urti come casuali. Il sistema diventa ergodico, quindi la traiettoria passa per tutti i punti dello spazio delle fasi. Cambiando di poco la configurazione iniziale del sistema si ottengono moti differenti.

Capitolo 2

Dinamica Molecolare

2.1 Go Model

I modelli Go [17] realizzati da Taketomi, Ueda e Go sono utilizzati in un'ampia gamma di simulazioni biomolecolari, in particolare per studiare il ripiegamento delle proteine. Sono modelli basati sulla struttura cristallografica della proteina (SBM), che si fondano sul principio di Anfinsen secondo il quale, allo stato nativo, le proteine assumono le interazioni locali ottimali.
La proteina, inizialmente in una conformazione casuale e disordinata, tramite la dinamica di Langevin ripiega nello stato nativo. Affinché la proteina transisca, nel modello di Go originale è stato tenuto in considerazione che gli amminoacidi hanno un'energia di interazione negativa solamente se sono in contatto, altrimenti non presentano interazione. Inoltre è possibile introdurre un'energia negativa solo quando l'angolo tra due amminoacidi è uguale a quello della struttura nativa predefinita.
Nel lavoro eseguito da Go ed i suoi collaboratori sono stati ottenuti risultati qualitativamente coerenti con gli esperimenti, infatti la proteina ripiega al di sotto della temperatura di folding e si dispiega al di sopra mostrando una transizione di fase del primo ordine alla temperatura critica.

2.2 SMOG

Per simulare la dinamica delle proteine con l'aiuto di un calcolatore l'utilizzo di hamiltoniane semplificate, come nel modello di Go, è un metodo molto efficace per ottenere moti di lunga durata e su larga scala. I modelli teorici più utilizzati per studiare la dinamica della proteine sono basati sull'ipotesi di un landscape energetico a forma di imbuto per il passaggio da stato denaturato a stato nativo. SMOG [18][19][20] è un programma che ha come obiettivo quello di generare una Go Model per una proteina [21] [22].
Per ottenere la struttura della proteina per il Go Model si utilizzano parallelamente approcci sperimentali e computazionali. Simulazioni di dinamica molecolare eseguite tramite un calcolatore permettono di indagare a livello microscopico la dinamica delle proteine e confrontare i risultati degli esperimenti dinamici con

quelli strutturali statici con risoluzione atomica.
Il modello *All-atom* include tutti gli atomi pesanti e non tiene conto degli atomi di idrogeno e tutti i parametri geometrici del sistema sono definiti dal file di input *.pdb*. Grazie a questo file di struttura si può minimizzare l'energia del sistema dal momento che, per costruzione, la configurazione nativa, cioè ad energia minima, è quella fornita nel file *.pdb*.

Per avviare la creazione del modello con SMOG è necessario fornire un input un file di struttura *.pdb* dove sono presenti solamente gli atomi e la separazione tra diverse catene. Dopo che il file di struttura è stato caricato il software restituirà come output i file necessari per implementare l'SBM su GROMACS.

Mentre nei modelli *coarse-grained* la forza di interazione complessiva tra i residui è omogenea poichè ogni residuo è una perlina, nei modelli *All-atom*, poiché residui diversi possono avere un numero di atomi diverso, la forza di interazione complessiva è eterogenea. Questa eterogeneità ha un effetto minimo nel processo di ripiegamento del sistema se la proteina è globulare e di piccole dimensioni, ma comunque tenere conto di questo fattore può fornire risultati più in accordo con quelli sperimentali [23].

Il software, per creare i file di topologia *All-atom* da implementare su GROMACS, fa uso di sistemi hamiltoniani basati su strutture che sono stati precedentemente pubblicati e convalidati. La hamiltoniana *All-atom* V_{AA} ha una forma funzionale rappresentabile nel seguente modo:

$$V_{AA} = \sum_{legami} \varepsilon(r - r_0)^2 + \sum_{angoli} \varepsilon_\theta(\theta - \theta_0)^2 + \sum_{planari} \varepsilon_\chi(\chi - \chi_0)^2 + \sum_{\substack{spina \\ dorsale}} \varepsilon_{SD} F_D(\phi) + \sum_{\substack{catene \\ laterali}} \varepsilon_{CL} F_D(\phi) + \sum_{contatti} \Big[\varepsilon_C \Big(\frac{\sigma_{ij}}{r}\Big)^{12} - 2\Big(\frac{\sigma_{ij}}{r}\Big)^{6}\Big] + \sum_{non-contatti} \varepsilon_{NC} \Big(\frac{\sigma_{ij}}{r}\Big)^{12}$$

dove il potenziale diedro F_D è definito come:

$$F_D(\phi) = \Big[1 - cos(\phi - \phi_0)\Big] + \frac{1}{2}\Big[1 - cos\big(3(\phi - \phi_0)\big)\Big]$$

I sette termini della hamiltoniana si possono suddividere in due categorie: i termini locali, che servono a descrivere interazioni locali e mantengono la geometria del sistema, e i termini non locali, che includono effetti di volume escluso e interazioni secondarie.
I primi 5 termini della hamiltoniana sono interazioni di contatto che tengono conto di come la geometria del sistema genera interazioni tra gli atomi, più precisamente si studiano interazioni atomiche dovute a distanza, angoli, planari, spina dorsale e catene laterali. Gli ultimi due termini sono il potenziale di Lennard-Jones per gli atomi a contatto e il termine repulsivo dello stesso potenziale per gli atomi non

in contatto.

Per definire il contatto tra due residui si osservano tutte le coppie di atomi condivisi e si controlla che siano in contatto. Ci sono diverse definizioni di contatto nativo, la più standard è quella usata nell'algoritmo *Shadow* che considera tutte le coppie di atomi in un raggio di 6Å [24]. Utilizzando questo tipo di algoritmo vengono introdotti diversi contatti "non fisici" tra atomi che pur trovandosi vicini tra loro sono schermati da contatti più interni. Per evitare questo si introduce la *Shadow contact map* [25]. Una sorgente luminosa è posta al centro dell'$i-esimo$ atomo, tutti gli atomi che hanno un'ombra proiettata su di loro vengono scartati come mostrato in fig. 2.1.
Dopo aver caricato il file *.pdb* e avviato il software, questo restituirà:

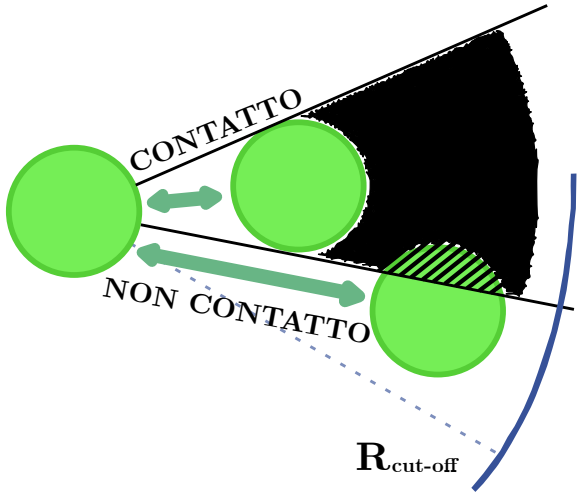


Figura 2.1: Rappresentazione dell'algoritmo *Shadow contact map*, che entro il raggio di cut-off crea dei contatti, a condizione che l'atomo non abbia delle ombre proiettate su di esso da altri atomi in contatto più vicini.

- **file .gro**: Struttura iniziale del file PDB centrato nel box.
- **file .top**: Descrive le interazioni atomiche nella hamiltoniana.
- **file .index**: Utile per strutture con tante catene.
- **mappa di contatto nativa**: Se selezionato l'algoritmo *Shadow*.
- **.outupt**: Contiene tutti i non-fatal warning e i messaggi.

Ottenuti questi file si può avviare una simulazione di dinamica molecolare prestando attenzione ad utilizzare il sistema di unità di misura adeguato per GROMACS:

- Massa: u (unità di massa atomica unificata, $1.660\,538\,86(28) \times 10^{-27}\,\mathrm{kg}$)
- Lunghezza: nm (nanometro, $10^{-9}\,\mathrm{m}$)

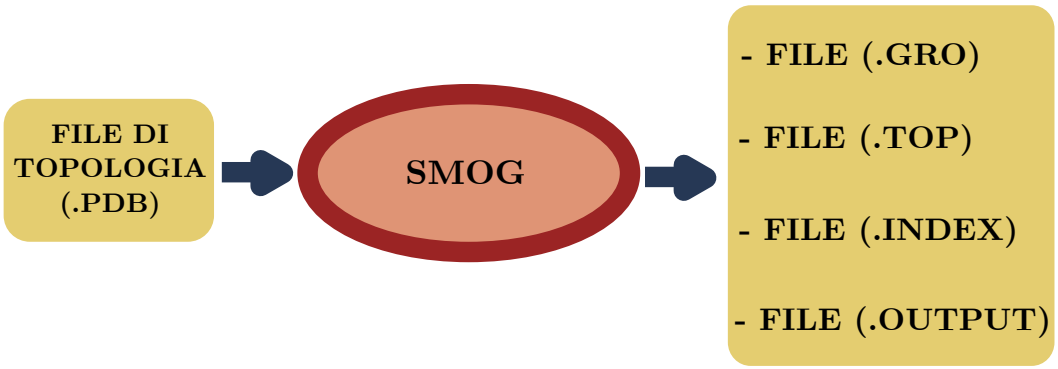


Figura 2.2: Schema riassuntivo dell'utilizzo di SMOG a cui viene fornito in input un file di topologia di una struttura biologica e restituisce i file di interesse per avviare una simulazione di dinamica molecolare

- Tempo: ps (picosecondo, 10^{-12} s)
- Carica: e (carica elementare, $1.602\,176\,53(14) \times 10^{-19}$ C)
- Energia: $\frac{kJ}{mol}$ (chiloJoule per mole, $1.660\,538\,86(28) \times 10^{-21}$ J)
- Forza: $\frac{kJ\,mol^{-1}\,nm^{-1}}{(1.660\,538\,86(28)\times 10^{-12}\,\mathrm{N})}$
- Pressione: $\frac{kJ\,mol^{-1}\,nm^{-3}}{(1.660\,538\,86(28)\times 10^{6}\,\mathrm{Pa})}$ (16.6 bar)
- Fattore elettrico: fel $\approx 138.935\,457(12)\,\frac{kJ\,mol^{-1}\,nm}{e^2}$, come in V $\approx \frac{\mathrm{fel}\,q^2}{r}$
- Potenziale elettrico: $\frac{kJ\,mol^{-1}\,e^{-1}}{0.010\,364\,269(1)\,\mathrm{Volt}}$, come in $\frac{\mathrm{fel}\,q}{r}$
- Costante di Boltzmann: $K_B \approx 0.008314472\,\frac{kJ}{molK}$

2.3 GROMACS

GROMACS [26][27] è un software per le simulazioni dinamiche di proteine, lipidi e acidi nucleici; l'obiettivo del software è quello di fornire un programma di dinamica molecolare versatile, efficiente e con codice sorgente [28]. Nei primi anni '90 il linguaggio di programmazione per l'ambito scientifico maggiormente utilizzato era *Fortran* per motivi di prestazione ed ottimizzazione. Sebbene GROMACS sia nato in quel periodo storico, è stato scritto in C, per renderlo più versatile e facilitare l'interazione con librerie di comunicazione, sviluppando una implementazione parallela portatile ed efficiente [29][30].
Nel 1995, per migliorare le prestazioni del software, nell'algoritmo sono stati isolati cicli interni che erano particolarmente importanti per la velocità di calcolo e sono stati riscritti in Fortran.

GROMACS fornisce diversi metodi per descrivere ed integrare il sistema [31]: la meccanica hamiltoniana microcanonica, dinamica stocastica, tra cui dinamica di Langevin e browniana, e metodi per la minimizzazione dell'energia.
Sono inclusi vari metodi per simulare bagni termici, in modo tale da poter mantenere la temperatura costante durante la simulazione. Nella descrizione del sistema GROMACS utilizza una scatola come cella fondamentale con condizioni al contorno periodiche. Una scatola triclinica è un tipo di cella unitaria utilizzata nelle simulazioni di dinamica molecolare per rappresentare la disposizione spaziale degli atomi.
In una scatola triclinica, i tre assi principali (a, b, c) non sono necessariamente perpendicolari tra loro e le loro lunghezze possono essere diverse. Gli angoli tra gli assi (α, β, γ) non sono necessariamente uguali a 90°. Questa flessibilità nella forma della cella unitaria permette di rappresentare una gamma più ampia di strutture rispetto a celle unitarie con simmetrie più restrittive.
Il tipo di interazione che si vuole utilizzare tra le molecole, cioè il campo di forza, è esterno a GROMACS e va implementato rispettando però i limiti posti dal software per il modo in cui organizza le valutazioni di forze ed energie.
L'integrazione dinamica segue la discretizzazione Leapfrog [32], che è reversibile e simplettica. Queste proprietà sono importanti per la stabilità dell'algoritmo. Un algoritmo è simplettico quando conserva area e volume dello spazio delle fasi.
L'algoritmo Leapfrog è un metodo numerico comunemente utilizzato per l'integrazione numerica delle equazioni differenziali ordinarie. L'idea principale dell'algoritmo è di suddividere il passo di integrazione in intervalli più piccoli e di aggiornare la posizione e la velocità del sistema in maniera alternata a metà passo. Questo metodo conserva la simmetria temporale e l'energia del sistema, rendendolo particolarmente utile per simulazioni a lungo termine.
Per processi stocastici come il folding proteico Sviluppati i file di topologia con SMOG si esegue la simulazione di dinamica molecolare con GROMACS in due passaggi. Innanzitutto si fissano i parametri per la simulazione tramite un file *.mdp* dove vengono specificati tutti i parametri di cui si vuole tenere conto durante la simulazione di dinamica molecolare.

I parametri sono numerosi, i più importanti sono:

- `integrator`: è il tipo di integratore con cui risolvere la dinamica del sistema.
- `dt`: è il passo temporale con cui integrare il sistema.
- `nstep`: numero di integrazioni che vengono eseguite.
- `nstxtcout`: serve per formattare l'output, in particolare fissa la frequenza con cui acquisire i frame della traiettoria.
- `pbc`: fissa le condizioni periodiche al contorno lungo le direzioni xyz.
- `table-extension`: estensione della tabella di ricerca del potenziale non legato oltre la massima distanza di cut-off. Potrebbe essere necessaria una lunghezza maggiore della tabella per le interazioni di Van der Waals e coulombiane.
- `cutoff-scheme`: genera un elenco di coppie con buffering.
- `nstlist`: frequenza con cui aggiornare la lista dei vicini, lista che GROMACS crea per calcolare le interazioni tra le diverse celle.
- `ns_type`: se impostato su *grid* crea una griglia nella casella e controlla solo gli atomi nelle celle della griglia vicine. Si costruisce un nuovo elenco dei vicini ogni passo di *nstlist*.
- `rlist`: distanza di cut-off per le interazioni a corto raggio.
- `coulombtype`: modo con cui calcolare le interazioni elettrostatiche. Se impostato su cut-off le calcola solo fino ad una certa distanza.
- `rcoulomb`: cut-off delle interazioni elettrostatiche.
- `vdwtype`: modo con cui calcolare le interazioni di Van der Waals. Se impostato su cut-off le calcola solo fino ad una certa distanza.
- `rvdw`: cut-off delle interazioni di Van der Waals.
- `tc-grps`: gruppo sul quale calcolare le interazioni dovute alla temperatura.
- `tau-t`: costante per la trasmissione del calore. Più è piccola più sono forti le interazioni.
- `ref-t`: temperatura del sistema.
- `pcoupl`: interazioni dovute alla pressione.
- `gen_vel`: generazione della velocità.
- `gen_temp`: temperatura per la distribuzione di Maxwell-Boltzmann.

Realizzato il file *.mdp*, si legge il file di topologia molecolare e si espande la topologia del sistema da una descrizione molecolare ad una atomica. Il comando `GROMPP` viene utilizzato per questo scopo e per leggere i parametri presenti nel file *.mdp* e tradurli in un file binario che si può utilizzare per avviare la simulazione di dinamica molecolare.
Il comando `MDRUN` è il motore principale per le simulazioni in GROMACS: legge il file binario di input generato precedentemente e produce diversi file che forniscono l'evoluzione temporale di energia, topologia, pressione e temperatura del sistema simulato.
Tramite il comando `MDMAT` è possibile creare matrici di contatto costituite dalla distanza minima tra coppie di residui di diversi amminoacidi.

Figura 2.3: Schema della sequenza di comandi e file da utilizzare per eseguire una simulazione di dinamica molecolare. Inizialmente si utilizzano alcuni file generati precedentemente da SMOG, da questi si genera il file *.tpr* tramite il comando `GROMPP` e infine si genera il file di traiettoria *xtc* tramite il comando `MDRUN`.

Capitolo 3

Reti Neurali

3.1 Machine Learning

"**Intelligenza Artificiale**" [33] è un termine attribuito ad azioni informatiche che imitano le funzioni cognitive tipicamente dell'uomo come l'apprendimento e la risoluzione di problemi. Il campo di ricerca dell'Intelligenza Artificiale (AI) viene definito come lo studio degli agenti intelligenti, dove per agente intelligente si intende qualsiasi dispositivo che nel suo ambiente compie azioni per massimizzare le possibilità di successo verso un obiettivo.
L'AI ad oggi è applicata in svariati ambiti, dalla comprensione del linguaggio umano alla risoluzione di sistemi di gioco strategici. I problemi centrali su cui si concentra l'AI comprendono la pianificazione, l'apprendimento, l'elaborazione del linguaggio, la percezione e la capacità di muovere e manipolare oggetti.
Il **Machine Learning** [34][35] è il sottocampo dell'AI che si concentra sulla creazione di un sistema software che possa apprendere o migliorare le performance in base ai dati che consuma. Questo è solo uno dei modi in cui si può generare una intelligenza artificiale.
Gli algoritmi di Machine Learning sono in grado di apprendere e predire strutture nei dati dopo aver realizzato un modello costruito a partire da un dataset iniziale. Questi modelli consentono a ricercatori e data analyst di prendere decisioni e produrre risultati affidabili, inoltre, attraverso l'apprendimento di relazioni e tendenze nei dati, viene favorita la nascita di nuove intuizioni.
I metodi di Machine Learning si suddividono in due macro-categorie, il Supervised Learning e l'Unsupervised Learning.

- Il *Supervised Machine Learning* utilizza dati etichettati, cioè dati organizzati in modo che ad ogni input della rete è associato un output. L'algoritmo di apprendimento riceve una serie di dati come input e impara comparando l'output calcolato con l'output corretto fornito dal dataset. Confrontando i due output la macchina è in grado di individuare errori nel processo di apprendimento e di modificare il modello di conseguenza.
- L'*Unsupervised Learning* invece, non utilizza dati tali per cui ad ogni input è anche associato l'output corretto che l'algoritmo deve imparare a prevedere.

> Poiché all'algoritmo di ML non viene fornita la risposta giusta, è esso stesso che deve ricercare dei pattern o una struttura all'interno del dataset per cercare di trarre delle conclusioni o cercare di capire cosa gli viene mostrato.

Il **Deep Learning** è lo studio delle *reti neurali artificiali* (ANN), algoritmi costituiti da una serie di strati formati da più unità di elaborazione non lineare che ricevono il segnale in ingresso tramite una somma pesata, lo prcessano tramite una trasformazione non lineare ed emettono a loro volta un segnale in uscita. Gli algoritmi di Deep Learning contengono più di uno strato nascosto e per questo la rete viene detta profonda. Una importate tipologia di ANN sono le *feed forward neural network* (FFNN) che in ogni strato utilizzano come input l'output dello strato precedente.
In una rete profonda ci sono molti strati tra l'input e l'output: questo consente all'algoritmo di compiere più passaggi di elaborazione composti da più trasformazioni lineari e non lineari.
Proseguendo lungo gli strati, le caratteristiche di livello superiore sono derivate da quelle di livello inferiore, in questo modo si viene formare una rappresentazione gerarchica dell'output.
In sintesi Intelligenza Artificiale, Machine Learning e Deep Learning sono algoritmi che hanno la capacità di decifrare i dati ispirandosi ai comportamenti umani. I due meccanismi principali con cui gli esseri umani interpretano la realtà sono: dare un nome a ciò che si osserva e riconoscere pattern, somiglianze o differenze tra input che si rivedono. I modelli di AI, ML, e DL cercano di riprodurre questi comportamenti.

3.2 Reti Neurali Feed Forward

Una rete neurale feed forward (FFNN) è un algoritmo di Deep Learning di ispirazione biologica formato da centinaia di singole unità (neuroni artificiali) collegati con coefficienti (pesi) che costituiscono la struttura neurale. Sono anche noti come elementi di elaborazione (PE) in quanto elaborano informazioni. Ogni PE ha ingressi ponderati, funzioni di attivazione e un output. Il PE è essenzialmente una funzione che processa input in output.
Una FFNN è in grado di fare previsioni accurate grazie alla sua struttura connessa dei neuroni artificiali che le permette di elaborare una grande quantità di dati. Ognuno di questi neuroni riceve un input ponderato dai pesi, lo elabora tramite una funzione di attivazione, che applica una trasformazione non lineare, ed emette un output.
Il tipo di connessione tra i neuroni è importante per il funzionamento del sistema. L'input di un neurone artificiale si traduce in un una somma pesata i cui termini sono positivi o negativi a seconda che il segnale sia inibitorio o eccitatorio:

$$\sum_{i=1}^{n} w_i x_i + \beta_0$$

dove n è il numero di neuroni dello strato precedente, w_i il peso dell'i-esimo neurone dello strato, x_i l'output dell'i-esimo neurone dello strato e β_0 un bias.

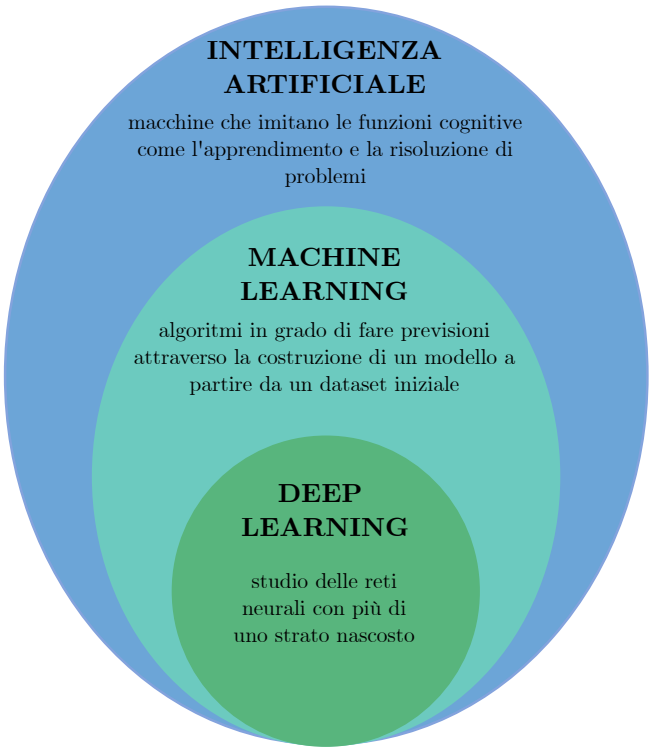


Figura 3.1: Tre definizioni dei termini Intelligenza Artificiale (AI), Machine Learning (ML) e Deep Learning (DL). Mentre un'AI è una qualsiasi macchina che imita funzioni cognitive umane, i modelli di ML e DL richiedono l'esistenza di un set di dati iniziale e la realizzazione di un modello.

Il segnale in ingresso viene elaborato attraverso la funzione di attivazione per produrre un singolo output del neurone.

$$f\left(\sum_{i=1}^{n} w_i x_i + \beta_0\right)$$

Secondo il teorema di Cybenko [36] data una rete neurale con funzione di attivazione non lineare, con un layer abbastanza grande, si è in grado di modellizzare qualunque funzione.
Una rete neurale può apprendere in modo differente lo stesso dataset iniziale, infatti il processo di apprendimento secondo il quale viene costruito il modello può cambiare a seconda di tre diversi parametri: funzione di attivazione, l'architettura e la regola di apprendimento.

- *Funzione di attivazione*: la funzione attivazione determina l'output di un neurone dato l'input ponderato e un eventuale termine di bias. La funzione di attivazione introduce una non linearità nella rete, permettendo alle reti neurali di modellare relazioni complesse nei dati. Alcuni esempi comuni sono la sigmoide, la tangente iperbolica, la ReLU e la ELU, mostrate in Fig. 3.4.
- *La sua architettura*: l'architettura di una rete neurale si riferisce alla struttura, all'organizzazione dei neuroni e dei loro collegamenti all'interno della

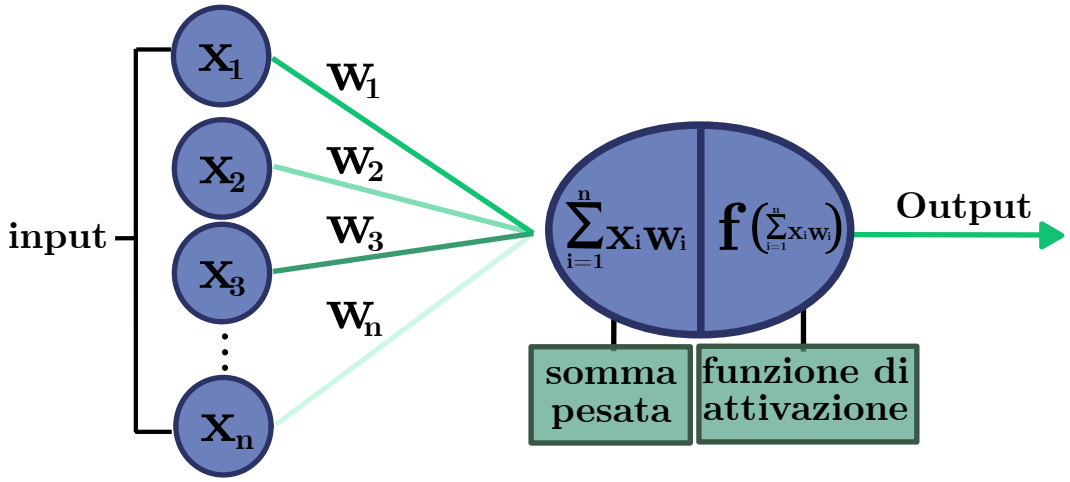


Figura 3.2: Immagine di un neurone artificiale che riceve n stimoli dallo strato precedente o dal dataset iniziale, processa gli stimoli in un unico input con una somma pesata, trasforma non linearmente la somma pesata tramite la funzione di attivazione ed emette un segnale in uscita per lo strato successivo.

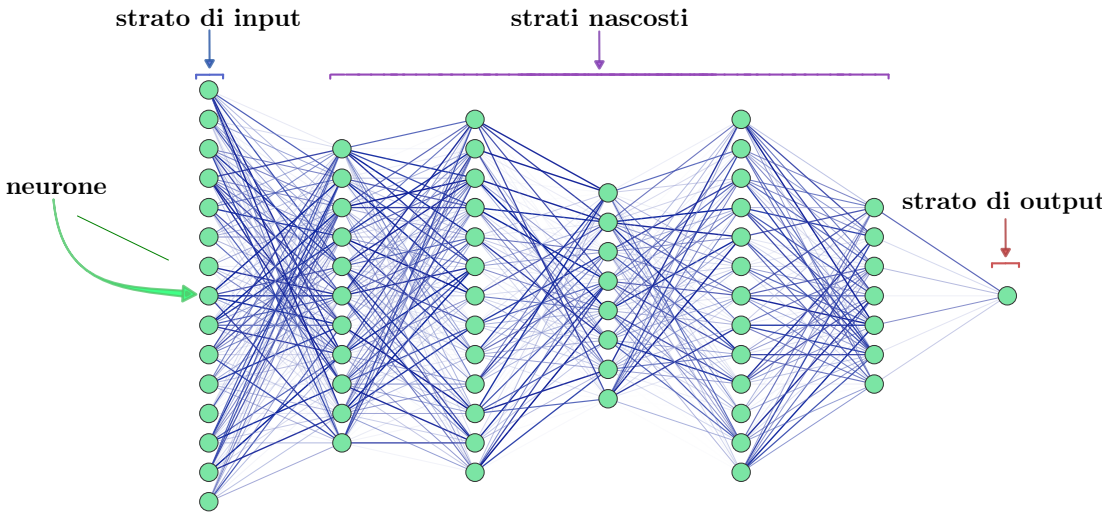


Figura 3.3: Rete neurale profonda dove sono rappresentati lo strato di input e output, gli strati nascosti e i neuroni. I contatti tra i neuroni possono avere intensità diversa.
I contatti più forti sono in blu scuro, i contatti più deboli sfumano verso il bianco.

rete, quindi al numero di neuroni e di strati. Questa struttura definisce come i dati vengono elaborati e trasformati attraverso la rete.

- *Regola di apprendimento*: la regola, o algoritmo, di apprendimento di una rete neurale definisce il criterio con cui la rete aggiorna i pesi iterativamente in modo tale che quest'ultima apprenda durante la fase di allenamento. L'algoritmo più usato per le FFNN è la retro-propagazione (backpropagation) che calcola l'errore di output della rete e lo propaga all'indietro attraverso l'architettura di quest'ultima per correggere i pesi dei collegamenti. Questo processo viene iterato numerose volte fino a quando l'errore non si è ridotto adeguatamente.

Durante l'addestramento i pesi vengono ottimizzati finché l'errore di previsione non viene minimizzato e la rete raggiunge il livello di accuratezza specificato. Una volta addestrata e testata, la rete può ricevere dati in ingresso per prevederne l'output. Dopo aver introdotto i meccanismi e i concetti fondamentali su cui

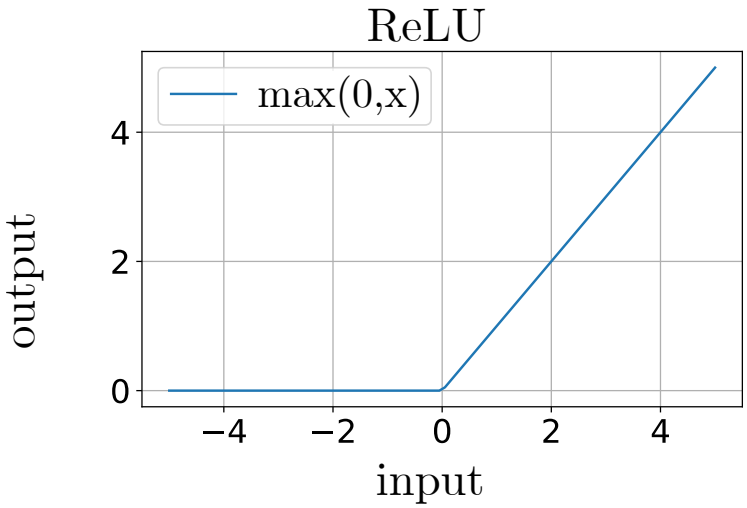


Figura 3.4: Esempio di funzione di attivazione che si ispira al modo in cui si attiva un neurone biologico e risulta computazionalmente efficiente.

si basa il Machine Learning, in appendice A si trattano in modo più rigoroso i metodi matematici e gli algoritmi che vengono utilizzati in questa branca dell'informatica [37].

3.3 PyTorch

PyTorch [38] è un modulo esterno del linguaggio Python [39] con diverse funzioni dedicate al Deep Learning ed al Machine Learning che permettono la costruzione di reti neurali profonde. Per manipolare i dati, PyTorch utilizza tensori ed offre anche un modulo `torch.utils.data` per la gestione del dataset iniziale con input ed output nei giusti formati.

Per realizzare una FFNN in PyTorch si crea un dataset di tensori coi dati a disposizione e si divide il dataset in due parti. La prima parte del dataset, solitamente più consistente (P.E. l'80%), viene utilizzata nella fase di addestramento del modello, mentre l'altra nella fase di test.
Dopo aver realizzato un dataset corretto si implementa l'architettura della FFNN formata da diversi strati lineari e non lineari utilizzando `torch.nn.Module` dove vengono definite due classi. Nella classe `__init__`, si definiscono gli strati della rete neurale, inclusi uno strato nascosto con una funzione di attivazione e uno strato di uscita. Nella classe `forward`, si definisce il flusso di dati attraverso la rete neurale tramite l'applicazione degli strati definiti nel metodo `__init__`. Un esempio di codice è mostrato in Fig. 3.5. I principali metodi e le principali classi

```
class Modello(nn.Module):
    def __init__(self, input_size, hidden_size, output):
        super(Modello, self).__init__()
        self.strato_nascosto = nn.Linear(input_size, hidden_size)
        self.relu = nn.ReLU()
        self.strato_uscita = nn.Linear(hidden_size, output)

    def forward(self, x):
        out = self.strato_nascosto(x)
        out = self.relu(out)
        out = self.strato_uscita(out)
        return out
```

Figura 3.5: modello di una FFNN realizzato con PyTorch formato da un singolo strato nascosto con funzione di attivazione ReLU.

su cui si basa un codice di Machine Learning in PyTorch sono:

- `torch.optim.SGD`: è una classe in PyTorch che implementa l'algoritmo di ottimizzazione della discesa del gradiente stocastico con cui si definisce un ottimizzatore.
- `nn.L1Loss`: è una classe che implementa la funzione di costo L1, nota anche funzione perdita della norma L1, definita matematicamente nell'equazione 4.1.
- `modello.train`: è un metodo che imposta il modello in modalità di addestramento. Questo metodo è utilizzato durante la fase di allenamento per segnalare a PyTorch che il modello deve essere configurato per calcolare i gradienti dei parametri durante la retropropagazione e aggiornare i parametri durante la fase di ottimizzazione.
- `modello.eval` è un metodo che imposta il modello in modalità di valutazione quindi non vengono più calcolati i gradienti o aggiornati i pesi.
- `ottimizzatore.zero_grad`: è un metodo utilizzato per azzerare i gradienti dei parametri del modello prima di eseguire una nuova retropropagazione.

Questo è necessario per evitare che i gradienti accumulati influenzino gli aggiornamenti dei parametri.

- `loss.backward`: è il metodo con cui si chiama la retropropagazione e si aggiornano i pesi.
- `ottimizzatore.step`: è un metodo che viene chiamato per eseguire l'aggiornamento dei parametri. L'ottimizzatore utilizza i gradienti e il tasso di apprendimento specificato per aggiornare i parametri in modo che la funzione costo del modello sia ridotta durante l'addestramento.

```
# Creazione del modello
modello = Modello()
# Definizione del criterio di loss e dell'ottimizzatore
criterio_loss = nn.CrossEntropyLoss()
ottimizzatore = optim.SGD(modello.parameters(), lr=0.01, momentum=0.9)
# Esempio di ciclo di addestramento e testing
epoche = 5
for epoca in range(epoche):
    # Fase di addestramento
    modello.train()  # Impostazione del modello in modalità di addestramento
    for batch, (data, target) in enumerate(train_loader):
        ottimizzatore.zero_grad()  # Azzeramento dei gradienti prima di ogni backward pass
        output = modello(data)  # Forward pass
        loss = criterio_loss(output, target)  # Calcolo della loss
        loss.backward()  # Backward pass
        ottimizzatore.step()  # Aggiornamento dei parametri
    modello.eval()  # Impostazione del modello in modalità di valutazione
    correct = 0
    total = 0
    with torch.no_grad():
        for data, target in test_loader:
            output = modello(data)  # Forward pass
            _, predicted = torch.max(output, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()
```

Figura 3.6: Esempio di codice PyTorch dove viene eseguita la fase di addestramento e di test del modello con tutti i metodi di Machine Learning citati.

In un modello di Machine Learning si dicono *iperparametri* i parametri che bisogna configurare prima di avviare l'addestramento.
A differenza dei parametri del modello, che vengono appresi durante l'addestramento stesso, gli iperparametri influenzano il comportamento e le prestazioni della rete, ma non vengono direttamente appresi dai dati.
Gli iperparametri più comuni sono:

- **Tasso di apprendimento** (*learning rate*): controlla la dimensione dei passi che l'algoritmo di ottimizzazione compie durante l'addestramento per aggiornare i pesi del modello. Un tasso di apprendimento troppo alto può far sì che l'ottimizzatore oscilli intorno al punto di minimo, mentre un tasso di apprendimento troppo basso può rallentare eccessivamente il processo di addestramento.
- **Numero di epoche**: Le epoche rappresentano il numero di volte in cui l'intero set di dati di addestramento processato dal modello durante l'ad-

destramento. Un numero troppo basso di epoche potrebbe non essere sufficiente per far convergere il modello, mentre un numero troppo alto potrebbe causare overfitting.

- **Dimensione del campionamento** (*batch size*): Indica il numero di input che vengono utilizzati in una singola iterazione durante il processo di addestramento. Una dimensione di campionamento più grande può accelerare il processo di addestramento, ma richiede più memoria, mentre una dimensione più piccola può migliorare la convergenza del modello, ma richiede più iterazioni.

- **Architettura del modello**: Gli iperparametri che definiscono l'architettura del modello includono il **numero di strati nascosti**, il **numero di neuroni in ciascuno strato** e la **funzione di attivazione**.

La scelta corretta degli iperparametri è cruciale per ottenere un modello di Machine Learning ben addestrato e performante. Gli iperparametri vengono spesso selezionati attraverso tecniche come la ricerca empirica o la ricerca casuale.

Capitolo 4

Metodi

4.1 Simulazioni di Dinamica Molecolare su proteina 1PGB

Lo scopo di questo progetto di tesi è di realizzare un modello in grado di apprendere e predire cambiamenti conformazionali in una proteina. Per fare ciò inizialmente è stato studiato il processo di ripiegamento della proteina 1PGB, composta da 56 amminoacidi, la cui struttura poco complessa la rende agile da simulare.
Inizialmente è stato scaricato il file di struttura (*.pdb*) della proteina 1PGB dalla *Protein Data Bank* [40], sono stati rimossi gli atomi di idrogeno e, utilizzando SMOG, è stato realizzato 1 SBM *All-atom* per studiare come varia l'RMSD in funzione della temperatura (fig. 4.1). In particolare, sono state eseguite 16 simulazioni di dinamica molecolare all'equilibrio della durata di 100 ps utilizzando la dinamica di Langevin e tenendo conto di interazione coulombiana e di Van der Waals con un raggio di cut-off di 1.5 nm, dove è stata fatta variare la temperatura del bagno termico da 50 $\frac{\text{Kmol}}{\text{kJ}}$ a 200 $\frac{\text{Kmol}}{\text{kJ}}$. Infine, per ognuna delle 16 simulazioni, è stato calcolato con GROMACS l'RMSD medio con il comando `RMS`.
Dal grafico ottenuto (Fig. 4.1), si osserva che l'andamento è quello di una sigmoide, come ci si aspetta per la transizione di fase del primo ordine, nel passaggio dallo stato Nativo compatto allo stato Denaturato disordinato superando la temperatura critica. La temperatura di folding è tra i 100 $\frac{\text{Kmol}}{\text{kJ}}$ e i 120 $\frac{\text{Kmol}}{\text{kJ}}$ e che l'RMSD (Eq. 1.3) allo stato nativo è circa 4 Å mentre allo stato denaturato aumenta a circa 16 Å.

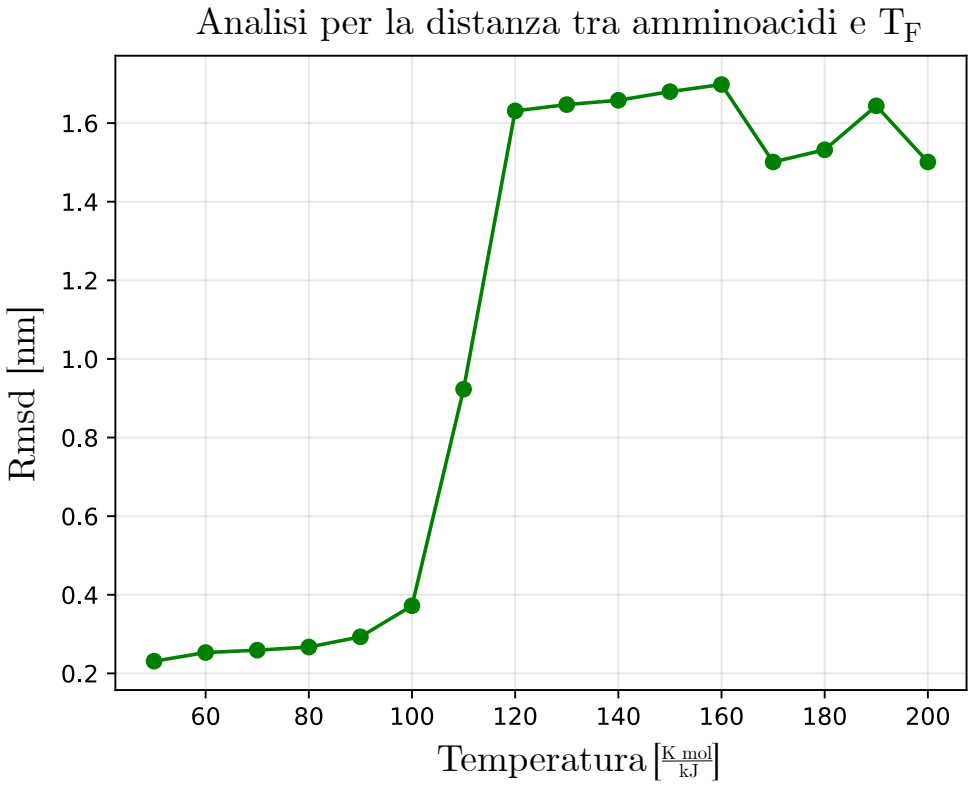
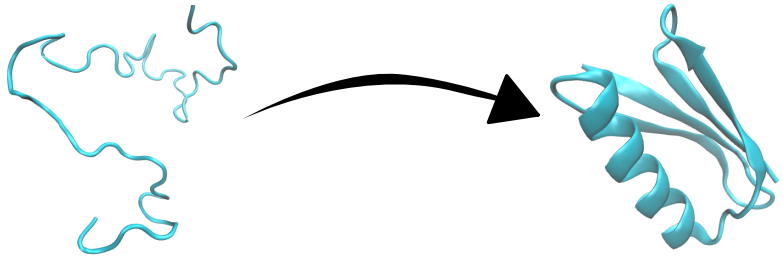


Figura 4.1: Studio dell'RMSD in funzione della temperatura nel quale si osserva una transizione di fase tra i 100 $\frac{\text{Kmol}}{\text{kJ}}$ e i 120 $\frac{\text{Kmol}}{\text{kJ}}$. Prima di superare la temperatura di folding i contatti distano meno di 4 Å, mentre allo stato denaturato circa 16 Å.

4.2 Traiettorie folding e mappe di contatto

Successivamente, con il comando `TRJCONV` di GROMACS, sono stati acquisiti 100 file di struttura differenti (*.gro*) dalla simulazione all'equilibrio a temperatura 200 $\frac{\text{Kmol}}{\text{kJ}}$ così che la proteina si trovasse allo stato denaturato. Dopo aver ottenuto i file, è stato possibile eseguire 100 simulazioni di traiettorie di folding da 500 ps, nelle quali la struttura iniziale era allo stato denaturato, ma la temperatura del bagno termico era 50 $\frac{\text{Kmol}}{\text{kJ}}$, tale per cui la proteina tende a transire verso lo stato nativo.
A causa della stocasticità del processo di ripiegamento, il tempo necessario a far ripiegare la proteina in ogni traiettoria non è noto a priori. Simulando 5 traiettorie di prova si è osservato che in circa 250 ps il processo di folding era terminato. Sebbene il tempo impiegato nelle simulazioni sia il doppio rispetto a quello osservato, non è sufficiente a garantire che in 500 ps la proteina ripieghi allo stato nativo in ognuna delle 100 traiettorie.
Dopo che, in virtù dall'analisi dell'RMSD svolta precedentemente, è stato impostato che due amminoacidi sono legati se la loro distanza è inferiore a 4 Å, sono state estratte 1000 mappe di contatto binarie da ognuna delle 100 traiettorie. In primo luogo, per estrarre le mappe binarie in formato *.xpm*, è stato utilizzato il



STATO DENATURATO STATO NATIVO

Figura 4.2: Visualizzazione della struttura tridimensionale dello stato nativo e dello stato denaturato per proteina 1PGB ottenute da due file *.gro*.

comando `MDMAT` di GROMACS; Successivamente, i file in formato *.xpm* sono stati convertiti in formato *.csv* grazie al pacchetto *DuIvyTools* [41].

A seguito di questa operazione si è in possesso di un dataset di mappe di contatto che può essere ottimizzato. Da ogni mappa è possibile eliminare i contatti superflui tra amminoacidi, infatti, ogni amminoacido è legato con sé stesso e con gli amminoacidi adiacenti a causa dei legami peptidici, inoltre, la mappa di contatto è simmetrica quindi ogni contatto è contato 2 volte.
Inoltre, da ogni traiettoria è possibile eliminare le mappe relative a simulazioni nelle quali la proteina non ha ripiegato o relative a traiettorie dove viene raggiunto lo stato nativo, ma che sono state acquisite dopo che si era già verificata la transizione di stato.
Sì è osservato dal file di struttura che, allo stato nativo, la proteina forma 125 contatti, sono quindi state eliminate le mappe relative a traiettorie che non raggiungono mai questo numero di contatti e che, quindi, non ripiegano. In aggiunta, sono state rimosse le mappe di traiettorie ripiegate, ma estratte a tempi successivi rispetto a quando erano già stati osservati tutti i contatti nativi.
In seguito a quanto fatto finora, per ogni traiettoria sono stati normalizzati i tempi relativi alle mappe di contatto estratte. Per normalizzarli, ogni tempo di ogni traiettoria è stato diviso per il tempo massimo della medesima, così da ottenere tutti valori compresi tra 0 e 1.
Infine, sono stati uniti tutti i file *.csv* rimanenti in un unico dataset in formato *.pth*, nel quale l'input è formato dalle mappe di contatto, mentre l'output è formato dai tempi veri della simulazione. In questo processo di pulizia e conversione le mappe di contatto sono state trasformate in tensori PyTorch di dimensione 1485 a cui è associato il tempo predetto, così da avere un dataset già pronto per essere caricato e processato in campionamenti durante l'allenamento della rete neurale.

4.3 Costruzione di FFNN per 1PGB

Dopo che il dataset è stato ripulito e normalizzato, si procede nella realizzazione di una FFNN che, fornitale in input una mappa di contatto, restituisce un tempo

normalizzato, che indica a che istante del processo di ripiegamento è associata la mappa di contatto.
Per realizzare la miglior FFNN è stato necessario cercare euristicamente i migliori iperparametri che minimizzassero la funzione costo. Poiché l'output della rete è monodimensionale si è utilizzata come funzione costo la *L1Loss* (eq. 4.1) di PyTorch, che è appunto la distanza indotta in uno spazio euclideo a norma 1:

$$L = \frac{1}{p} \sum_{i=1}^{p} |y_i - \hat{y}_i| \tag{4.1}$$

Dove p è la batch size, y_i il tempo predetto dal modello e $\hat{y}_i$ il tempo vero fornito dal dataset.
Per ottenere il miglior modello sono state eseguite numerose simulazioni, nelle quali sono stati impostati in modo differente: learning rate, batch size, numero di epoche, funzione di attivazione, numero di nodi e numero di strati, ed è stato osservato per quali iperparametri la funzione costo del training e testing dataset si riducessero al meglio, infatti, per realizzare il modello, l'80% del dataset iniziale è stato impiegato nella fase di trainig, mentre il restate 20% nella fase di testing. Poiché l'input della FFNN è ad elevata dimensionalità, mentre l'output è monodimensionale, si è cercato di realizzare un'architettura ad imbuto, cioè tale per cui ad ogni strato il numero dei nodi si riduce di un fattore 8. Il risultato finale è una rete di 4 strati composti rispettivamente da 1024, 128, 16 e 2 neuroni, con funzione di attivazione *ReLU*, batch size 32 e 3000 epoche.

4.4 Validazione del modello ed interpretazione dei risultati

Dopo aver ottenuto il modello migliore per la proteina 1PGB, si è cercato di valutare grazie a quali contatti e con quale criterio la rete neurale distingue le diverse mappe di contatto.
Poiché le deep neural network dipendono da un elevato numero di parametri, la funzione costo è estramente complessa. Si preferisce trattare il modello come una black-box, cioè come una funzione che non si è in grado di valutare dall'interno, e studiare statisticamente i risultati ottenuti e predetti cercando di comprenderne i meccanismi osservando quali tempi vengono predetti in funzione di particolari mappe di contatto.

Per prima cosa, è stata confrontata la distribuzione dei tempi veri con la distribuzione dei tempi predetti, per osservare se sono presenti presenti errori nel modo in cui la rete predice i dati o se invece il modello è preciso ed accurato.
In secondo luogo è stato realizzato un grafico a dispersione tra i tempi veri e i tempi predetti e, dopo essere stato effettuato un fit lineare tra i punti per studiare la correlazione dei dati, si è calcolando il coefficiente di Pearson.
Il modello supera ampiamente i test statistici applicati sul dataset, infatti, le distribuzioni dei tempi veri e predetti sono entrambe uniformi e la correlazione tra

i tempi è $R = 0.99$. Ciò non è sufficiente per affermare che il modello è corretto, infatti non è detto che ciò di cui la rete si è servita per apprendere siano effettivamente i contatti che si formano durante il processo di ripiegamento; potrebbe essere presente un bias che rende il modello errato nonostante l'accuratezza.

Dopo aver studiato la struttura tridimensionale della proteina 1PGB, è stato possibile effettuare un ulteriore test realizzando mappe di contatto con solo alcuni contatti ad hoc formati per la rete neurale ed osservando la coerenza tra gli output restituiti e quanto ci si aspetta. In particolare, sono state realizzate mappe di contatto dove si formano progressivamente i contatti della struttura secondaria e successivamente i contatti della struttura terziaria della proteina.
Osservando la struttura nativa della proteina e paragonandola coi contatti nella mappa nativa, si nota che la struttura secondaria, composta da due β-forcine ed un'α-elica, è rappresentata sulla mappa tramite dei contatti diagonali mentre la struttura terziaria da altri contatti più complessi che rappresentano i contatti tra le strutture secondarie sopracitate, come mostrato in fig. 4.3.

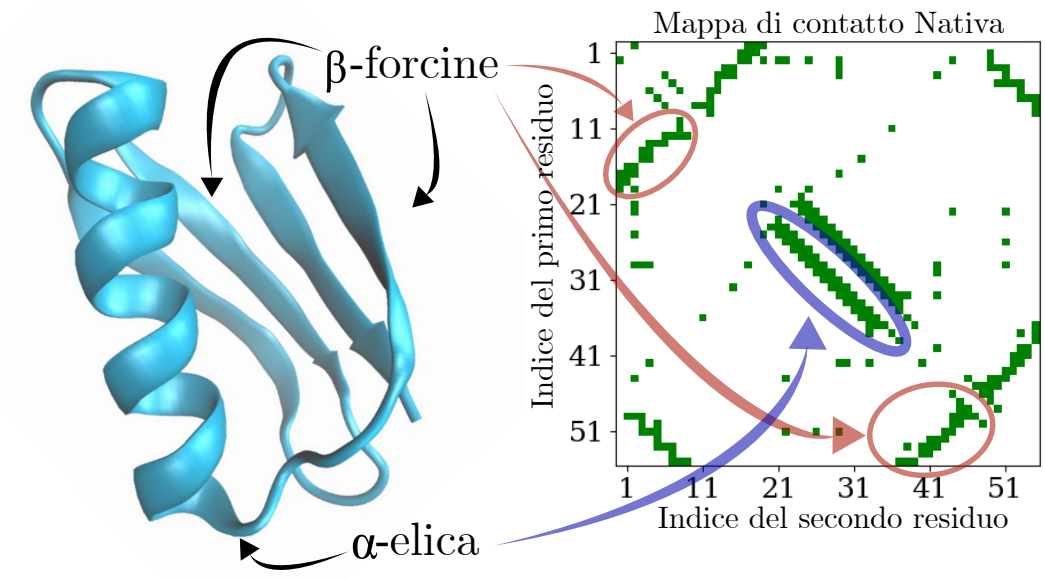


Figura 4.3: In questa immagine sono mostrate le analogie tra la struttura tridimensionale della proteina 1PGB allo stato nativo e la relativa mappa dei contatti. In particolare si osserva che la struttura secondaria, formata da 2 β-forcine ed un'α-elica, si ritrova nella mappa dei contatti tramite una concentrazione di contatti in prossimità degli angoli in altro a sinistra ed in basso a destra e lungo la diagonale.

É stata decomposta la struttura nativa della proteina 1PGB in 7 sotto-strutture denominate: β-forcina alta, β-forcina bassa, α-elica, rettangolo-basso, rettangolo-alto, diagonale ed unione-forcine, come mostrato in Fig. 4.4.
Costruite le mappe di contatto con solamente la sotto-struttura desiderata, è stata realizzata una distribuzione di 300 mappe di contatto alle quali è stato ag-

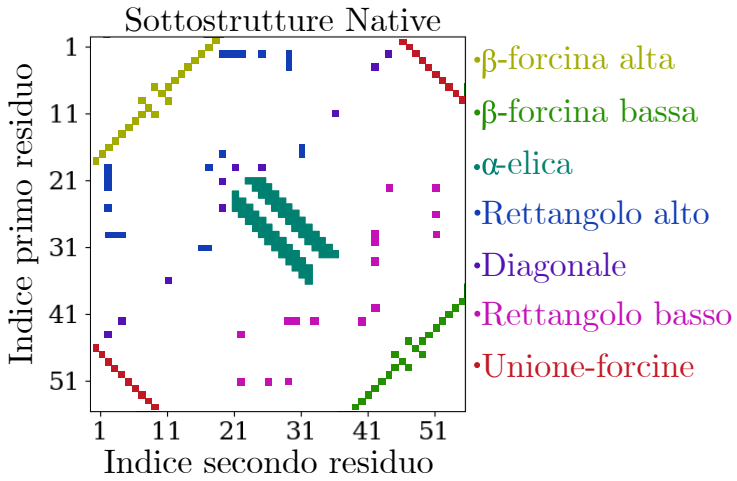


Figura 4.4: Decomposizione dei legami presenti nella mappa dei contatti della proteina 1PGB nello stato nativo in aclune sotto-strutture secondarie e terziarie.

giunto un rumore gaussiano nella generazione dei contatti adiacenti, così che il tempo predetto della rete neurale non dipenda da singoli contatti specifici ed isolati, ma dalla globalità della struttura della mappa di contatto.
Dopo che sono state effettuate diverse predizioni per permutazioni differenti e combinazioni lineari delle sotto-strutture, si è osservato che, combinando linearmente le sette distribuzioni, partendo da una sola sotto-struttura ed aggiungendone progressivamente altre, sono state ottenute delle distribuzioni di tempi che aumentano progressivamente da 0 a 1 man mano che si raggiunge la struttura nativa.
Una di queste combinazioni coincide con la formazione iniziale della struttura secondaria realizzata dalle due β-forcine e dall' infine dall'α-elica. Dopo che si è formata interamente la struttura secondaria, si forma la struttura terziaria con rettangolo-alto, diagonale, rettangolo-basso ed infine unione-forcine.
Una sequenza di questo tipo durante il processo di folding è coerente con quanto si osserva nelle simulazioni effettuate delle traiettorie di ripiegamento; infatti, inizialmente si tendono a formare l'α-elica e le β-forcine, in seguito le β-forcine e l'α-elica si legano tra loro.

Sebbene efficace, il metodo utilizzato non è facilmente generalizzabile a proteine più complesse poiché la loro struttura secondaria e terziaria può essere complessa e difficile da interpretare. Inoltre, con una analisi di questo tipo, non è chiaro quali contatti tendono a formarsi prima e quali dopo durante il processo di ripiegamento, poiché la decomposizione è grossolana ed eseguita manualmente, senza rispettare un metodo riproducibile da un calcolatore.

Per ovviare queste limitazioni si è cercato "invertire la rete neurale", cioè di creare un algoritmo che, sfruttando il modello della FFNN, tramite un processo di ottimizzazione stocastica, restituisce come output una probabile mappa di contatto in relazione al tempo fornito come input all'algoritmo.
Più precisamente l'algoritmo inizialmente genera casualmente una mappa di con-

tatto a partire da alcune condizioni iniziali e, definita una funzione costo calcolata come il modulo della differenza tra il tempo desiderato e il tempo predetto dalla mappa generata, si modifica la mappa eseguendo la SGD e la retropropagazione per minimizzare la funzione costo.
La SGD è un processo di ottimizzazione continuo e per questo tende a formare i contatti in modo continuo associando al legame anche un peso, mentre le mappe di contatto sono binarie. Per questo, si è cercato di modificare la funzione costo aggiungendo una penalizzazione nel caso in cui il valore dei contatti sia intermedio tra 0 e 1, negativo o superiore ad 1, così che siano favoriti valori dei contatti binari. Inoltre, tramite questo algoritmo, è possibile generare numerose distribuzioni di mappe di contatto differenti al variare delle condizioni iniziali, poiché il processo di ottimizzazione è stocastico. Ottenute 10 distribuzioni diverse da 500 mappe di contatto ciascuna, relative ai 10 tempi equispaziati tra 0.1 e 1, sono stati calcolati i contatti più frequenti ottenuti e paragonati coi contatti nativi.
Per ottenere i contatti più frequenti e paragonare l'output ottenuto con la configurazione nativa, è stata osservata la distribuzione dei pesi normalizzati associati ai contatti. Come mostrato in Fig. 4.5 la distribuzione è bimodale e presenta due picchi in un intorno di 0.4 e di 0.6.
Risulta chiaro che i contatti non formati si distribuiscono attorno al primo picco

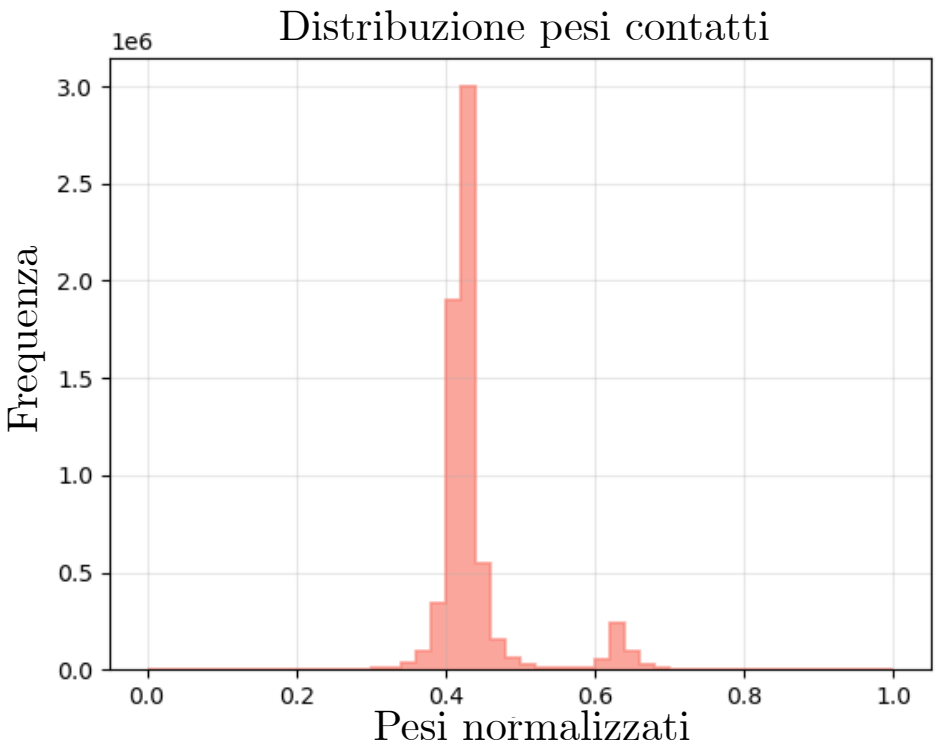


Figura 4.5: Distribuzione dei pesi normalizzati associati ai contatti formati dall'algoritmo di ottimizzazione stocastica relativi per tutti i tempi tra 0.1 e 1.

e sono molto più numerosi di quelli formati che si distribuiscono attorno al secondo picco. In questo modo è possibile definire un limite di 0.60 tra un contatto

formato ed uno no.
Definito che un contatto è formato se gli è associato un peso superiore a 0.60, è stato calcolato il tensore media dei pesi dei contatti per ogni tempo tra 0.1 e 1. Successivamente, il tensore media stato normalizzato sull'intervallo tra il minimo (0.1375) ed il massimo (0.568) dei pesi medi per i contatti dei diversi tempi. Infine, sono stati binarizzati i tensori media normalizzati.
Distinti i contatti formati e non nel tensore media binarizzato per ogni tempo, si sono confrontati i contatti dell'algoritmo di ottimizzazione coi contatti nativi.

4.5 Funzione costo e numero di traiettorie

Grazie a quanto fatto precedentemente, si è verificato che il modello sviluppato dalla rete neurale è accurato e in grado di distinguere le mappe di contatto in funzione dei contatti che si formano durante il processo di ripiegamento. In seguito si è studiato come varia l'errore in funzione del numero di traiettorie impiegate per l'addestramento della rete. Per fare ciò, si è allenato il modello con un numero di traiettorie sempre ridotto ed è stata calcolata la funzione costo di addestramento e di test. Come mostrato in fig. 4.6, si osserva che una singola traiettoria è sufficiente per allenare un modello con un errore nella predizione dei tempi del 14%.

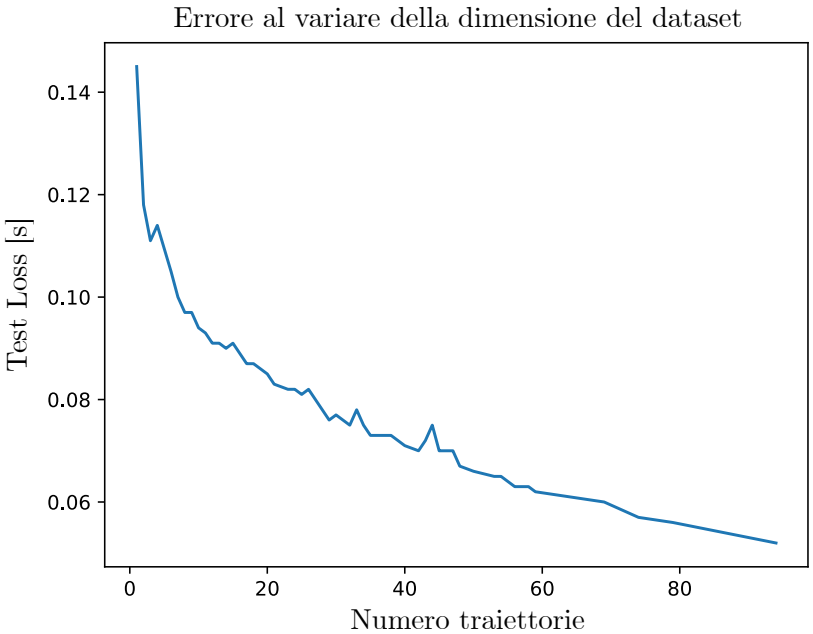


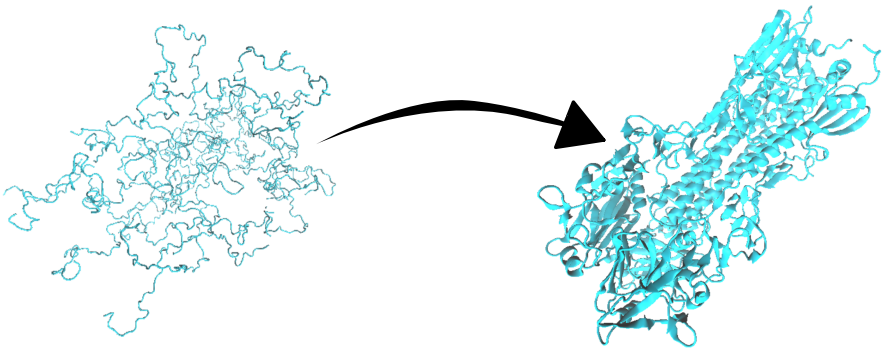
Figura 4.6: Studio dell'esito del valore che assume la funzione costo dopo essere minimizzata al variare del numero di traiettorie che allenano il modello. Si osserva che anche per una singola traiettorie l'errore commesso è del 14%.

4.6 Proteina 2HMG

Dopo aver studiato il modello approfonditamente per la proteina 1PGB si è verificato che anche per proteine più grandi e complesse la rete neurale sviluppata è in grado di predire il tempo di ripiegamento.
É stata utilizzata l'emoagglutinina (2HMG) [42], una glicoproteina virale presente sulla superficie del virus dell'influenza formata da 1510 amminoacidi. Si trova sotto forma di due omotrimeri, il che significa che è composta da 2 parti formate da tre sub-unità identiche. Ognuna delle due subunità di emoagglutinina è costituita da catene proteiche chiamate rispettivamente HA1 e HA2. Quindi, in totale, l'emoagglutinina è composta da sei catene proteiche (tre HA1 e tre HA2).

É stato scaricato il file di struttura *.pdb* dalla *Protein Data Bank*, si è fornito in input a SMOG tale file imponendo una scatola di 1000 nm e sono stati ottenuti i file *.gro* e *.top*.
La prima simulazione eseguita è ad alta temperatura (300 $\frac{\text{Kmol}}{\text{kJ}}$) con struttura iniziale della proteina allo stato nativo così che si transisca verso lo stato denaturato.

Dopo aver ottenuto, grazie al comando `TRJCONV` un file di struttura per lo stato denaturato, è stata eseguita una simulazione di dinamica molecolare con GROMACS per la emoagglutinina ad una temperatura di 10 $\frac{\text{Kmol}}{\text{kJ}}$ dove a partire dallo stato denaturato, in un tempo di 15 ns, viene raggiunto lo stato nativo, come mostrato in fig.4.7.



STATO DENATURATO STATO NATIVO

Figura 4.7: Rappresentazione dello stato denaturato e dello stato nativo della proteina emoagglutinina (2HMG). Lo stato iniziale e finale della simulazione a bassa temperatura eseguita su GROMACS.

Ottenuto il file *.xtc*, si estraggono 1000 mappe di contatto e si ripulisce il dataset ottenuto in modo analogo a quanto fatto con la proteina 1PGB, così da essere in possesso di un dataset formato da 1000 file *.csv* per la nuova proteina.
Realizzato il nuovo dataset per l'emoagglutinina, è possibile allenare nuovamente la FFNN, minimizzare la funzione costo ed ottenere un modello in grado di associare il tempo di folding normalizzato a mappe di contatto molto più complesse

di quelle della proteina 1PGB.
Impostando 200 epoche, funzione di attivazione *ReLU*, dimensione dei campionamenti 16, numero di strati 4 e numero di neuroni per ogni strato rispettivamente 2048, 256, 32, 4 è stato ottenuto il miglior modello per la proteina 2HMG.

Infine, è stato utilizzato l'algoritmo di ottimizzazione stocastica per studiare la formazione dei contatti nel processo di ripiegamento. Per fare ciò, sono state predette 10 mappe di contatto per ognuno dei 10 tempi compresi tra 0 e 1 equispaziati da unità di 0.1. É stata studiata la distribuzione dei contatti in modo analogo a quanto fatto per la proteina 1PGB. Come mostrato in Fig. 4.8, i due picchi sono in 0.5 e 0.9, quindi si sceglie come valore soglia 0.7. Successivamente, sono stati calcolati i tensori dei pesi medi per ogni tempo normalizzati sull'intero dataset e sono stati bianarizzati utilizzando come valore soglia 0.7, così da ottenere le mappe di contatto relative ad ogni tempo.

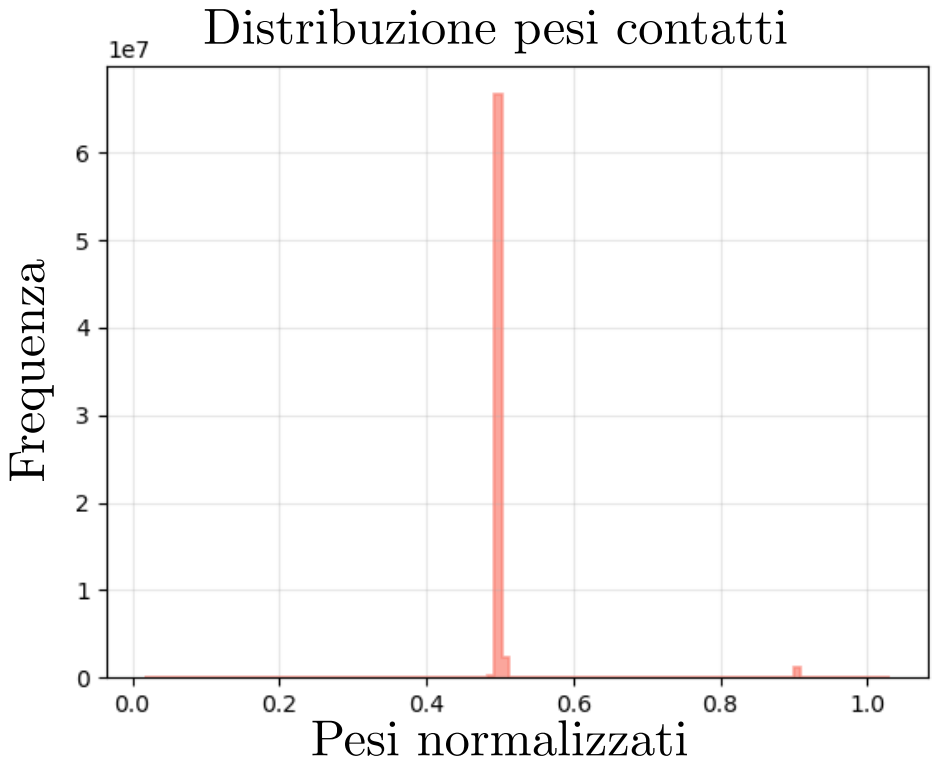


Figura 4.8: Distribuzione dei pesi normalizzati per la proteina 2HMG. Si osserva una distribuzione con due picchi localizzati in 0.5 e in 0.9.

Capitolo 5

Risultati

5.1 Risultati

In questo progetto di tesi è stato studiato il processo di ripiegamento delle proteine 1PGB e 2HMG con tecniche di Machine Learning, in particolare è stato sviluppato un modello che, fornita una mappa di contatto in input, restituisce il tempo di folding normalizzato. Per la 1PGB, tale modello è stato costruito tramite una FFNN composta da 4 strati rispettivamente di 1024, 128, 16 e 2 neuroni, con funzione di attivazione $ReLU$, batch size 32 e 3000 epoche dove la funzione costo calcolata sul train dataset (test loss) e sul test dataset (test loss) è stata minimizzata ed è stato ottenuto un errore tra il tempo predetto dal modello e quello vero del 0.7% per la *trainloss* e del 5.5% per la *testloss*, come mostrato in Fig. 5.1.

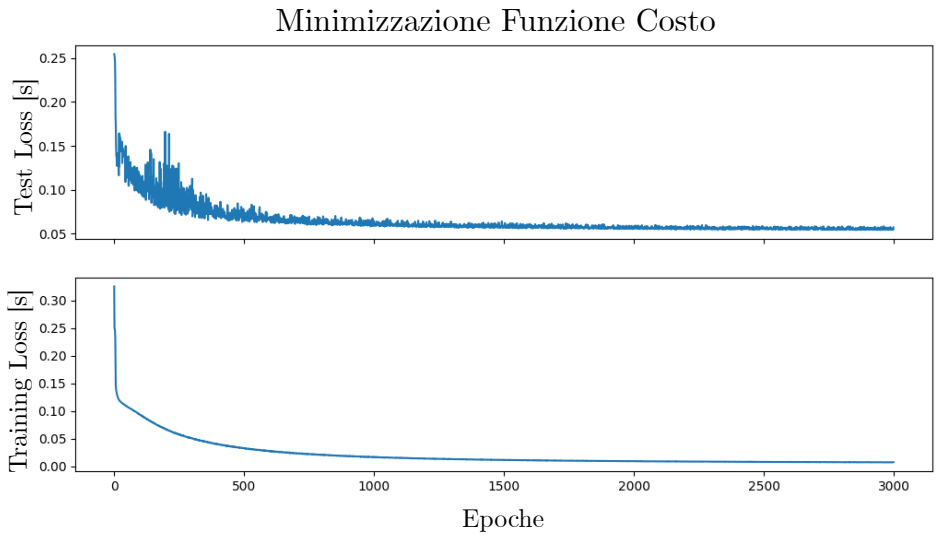


Figura 5.1: Andamento della funzione costo, quindi del differenza tra il tempo predetto dal modello ed il tempo vero, durante l'allentamento e durante la fase di collaudo del modello. La minima *train loss* = 0.007, mentre la minima *test loss* = 0.055.

Per testare la validità del modello, sono state confrontate le distribuzioni dei tempi predetti e dei tempi veri che risultano essere entrambe uniformi. In aggiunta, l'indice di correlazione di Pearson tra i tempi risulta essere $R = 0.99$, come mostrato in fig 5.2.

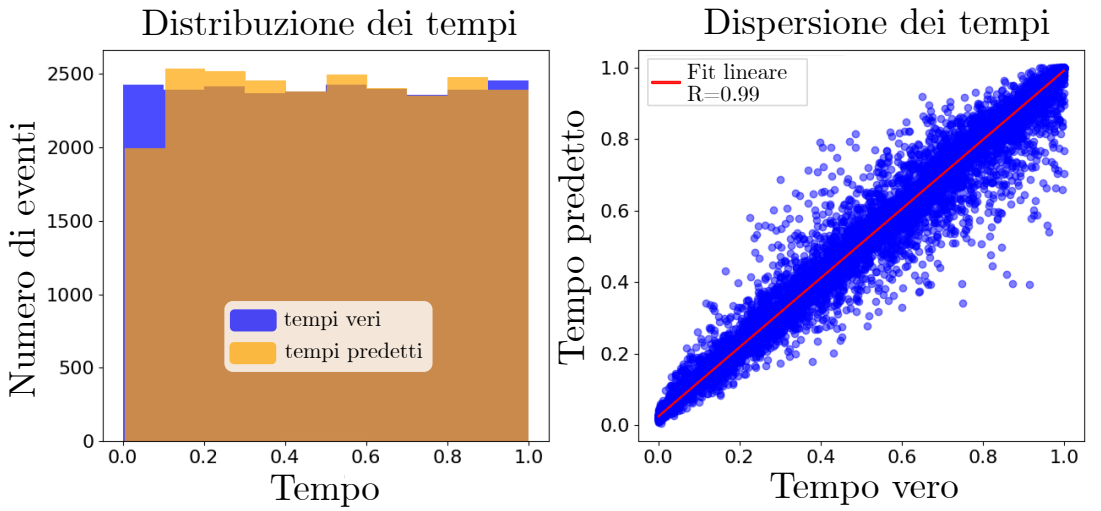


Figura 5.2: A sinistra, il confronto tra la distribuzione dei tempi veri presenti nel dataset e la distribuzione dei tempi predetti dal modello, che risultano essere entrambi uniformi.
A destra, la dispersione tra i tempi veri e i tempi predetti comparata al fit lineare, raffigurato in rosso, che è stato impiegato per il calcolo del coefficiente di correlazione R.

Oltre a verificare il modello tramite test statistici, è stato studiato come varia la distribuzione dei tempi predetti al formarsi della struttura tridimensionale della proteina 1PGB, come mostrato in Fig. 5.3.
Al formarsi della struttura tridimensionale della proteina 1PGB, il valor medio della distribuzione dei tempi risulta spostarsi progressivamente da 0 a 1, ciò significa che la proteina si ripiega formando le strutture individuate. Si osservi che le tre distribuzioni centrali risultano essere fortemente sovrapposte, ciò significa che, a causa della cooperatività del sistema, la formazione dell'α-elica, il legame tra l'α-elica e il β-foglietto alto e i primi legami tra i due β-foglietti iniziano a formasi contemporaneamente e può accadere che, per diverse traiettorie, si formino in ordine cronologico differente.

Validata la FFNN e compreso come ripiega la proteina 1PGB, è stato realizzato un algoritmo di ottimizzazione stocastica che, tramite la SGD, minimizza la differenza tra il tempo predetto dal modello associato alla mappa realizzata ed il tempo desiderato. Realizzate 500 mappe di contatto per ognuno dei dieci tempi tra 0 e 1, intervallati ogni 0.1 unità, è stata calcolata la media dei pesi per ogni contatto della proteina, sono stati selezionati i contatti più intensi studiando la distribuzione dei pesi associati ai contatti e sono state binarizzate le mappe di

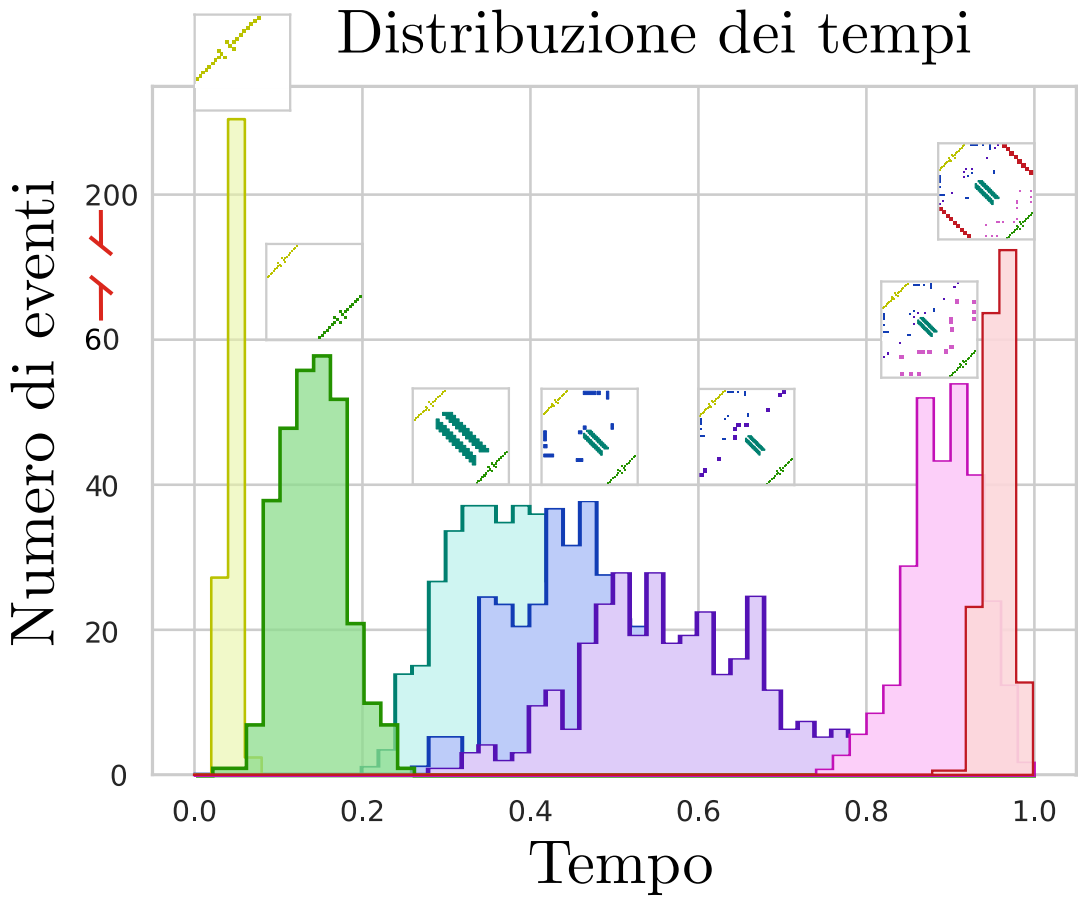


Figura 5.3: Distribuzioni dei tempi predetti dal modello relative a 7 dataset formati da 300 mappe di contatto con rumore gaussiano la cui struttura è mostrata in figura. Si osservi che al formarsi delle sottostrutture native il tempo predetto per la distribuzione aumenta progressivamente verso l'1.

contatto.
Si osserva che, all'aumentare del tempo verso l'1, i contatti più importanti sono coerenti con la struttura dei contatti nativi, come mostrato in Fig. 5.4. Anche i pesi mediati e normalizzati più grandi, all'aumentare del tempo, si trovano in prossimità delle aree in cui sono presenti le strutture native perché l'algoritmo tende a formare tali strutture.

Superati i test applicati, risulta evidente che il modello predice accuratamente i risultati per la proteina 1PGB e che l'algoritmo di ottimizzazione stocastica ricerca i contatti a tempi diversi coerentemente a quanto appreso dalla rete neurale.
Successivamente, si è cercato di allenare la rete neurale con una proteina molto più complessa: l'emoagglutinina (*2HMG*). Per fare ciò, innanzitutto, è stata studiata l'accuratezza del modello al variare del numero di traiettorie impiegate nell'allenamento. Dai risultati ottenuti è chiaro che anche una singola traiettoria è sufficiente per ottenere un modello con una incertezza del 15%, come mostrato

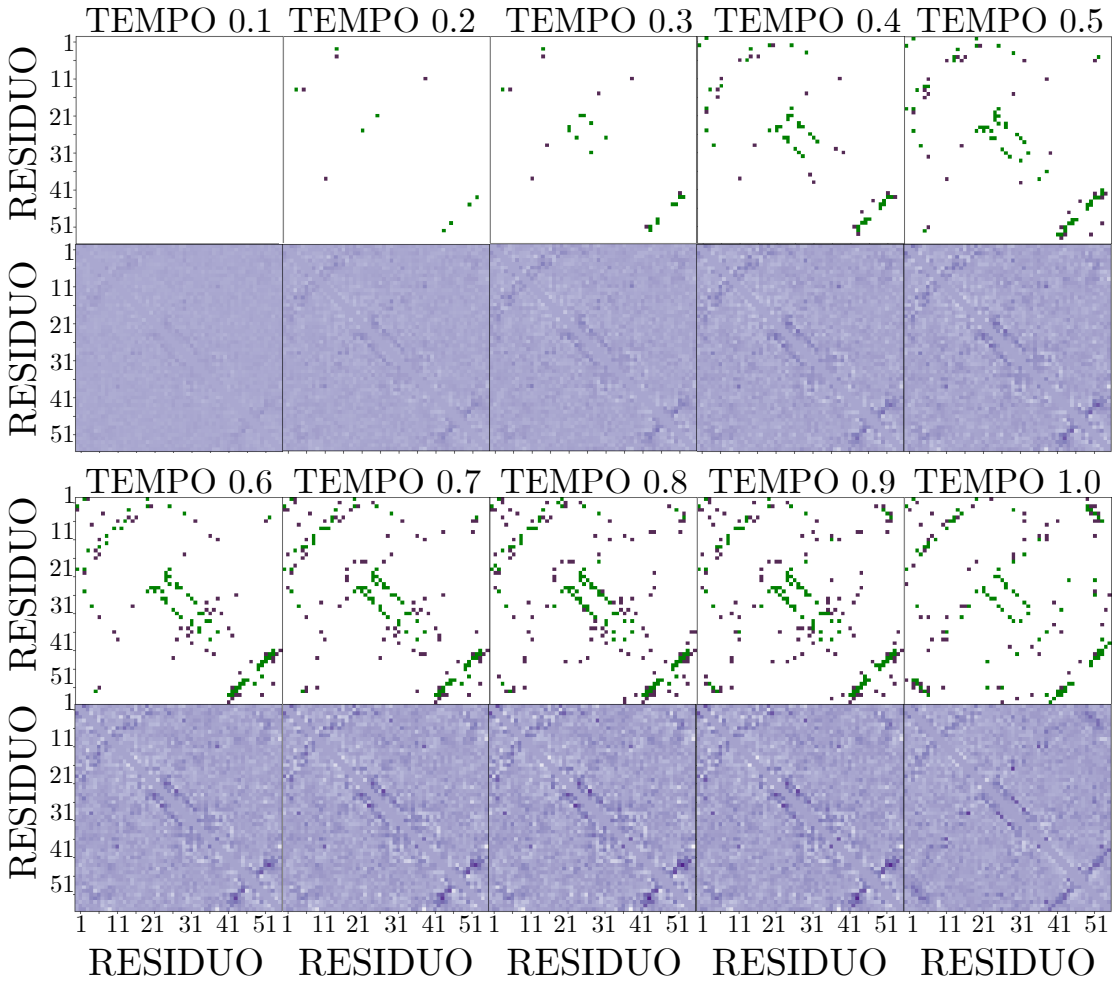


Figura 5.4: Mappe di contatto predette dall'algoritmo di ottimizzazione stocastica per i 10 tempi equispaziati tra 0.1 e 1.
Nelle mappe discrete, i contatti in viola sono i contatti ottenuti non nativi, mentre quelli verdi sono i contatti presenti anche alla conformazione nativa. Le mappe non discrete mostrano i pesi normalizzati dei diversi contatti all'aumentare del tempo.

in Fig. 4.6.
In seguito, è stata eseguita una simulazione di dinamica molecolare in cui la proteina 2HMG, inizialmente allo stato denaturato, raggiunge lo stato nativo. Grazie a tale simulazione, è possibile formare il dataset, composto da mappe di contatto e tempi normalizzati associati ad esse, per allenare la FFNN con i migliori iperparametri.
Con 200 epoche, funzione di attivazione *ReLU*, 4 strati formati rispettivamente da 2048, 256, 32, 4 neuroni e batch size di 16 è stato ottenuto il miglior modello con un errore durante la fase di allenamento dell'1.1%, come illustrato in Fig. 5.5.
Infine anche per la proteina 2HMG è stato impiegato un algoritmo di ottimizzazione stocastica analogo a quello impiegato per la proteina 1PGB per osservare la formazione della struttura tridimensionale al progredire del tempo. Le mappe di

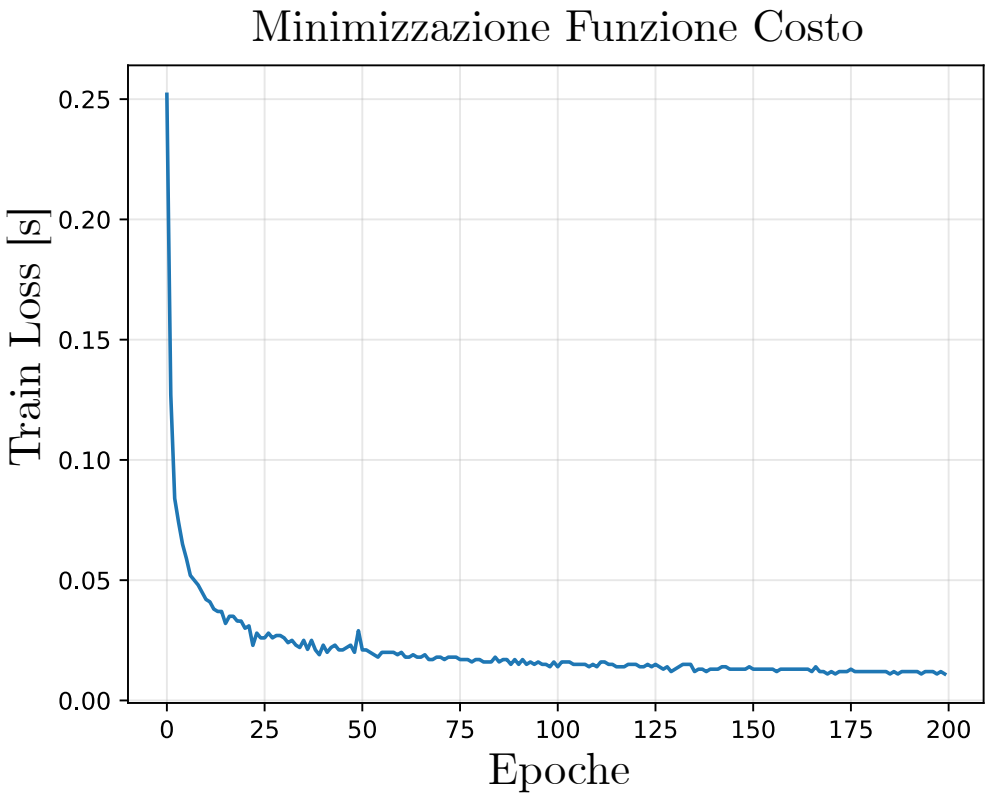


Figura 5.5: Minimizzazione della funzione costo durante la fase di addestramento del modello per la proteina 2HMG. La Loss minima è $Loss = 0.11$ che coincide con un errore in percentuale sul tempo predetto dell'1.1% sulla singola traiettoria.

contatto ottenute sono mostrate in Fig. 5.6.

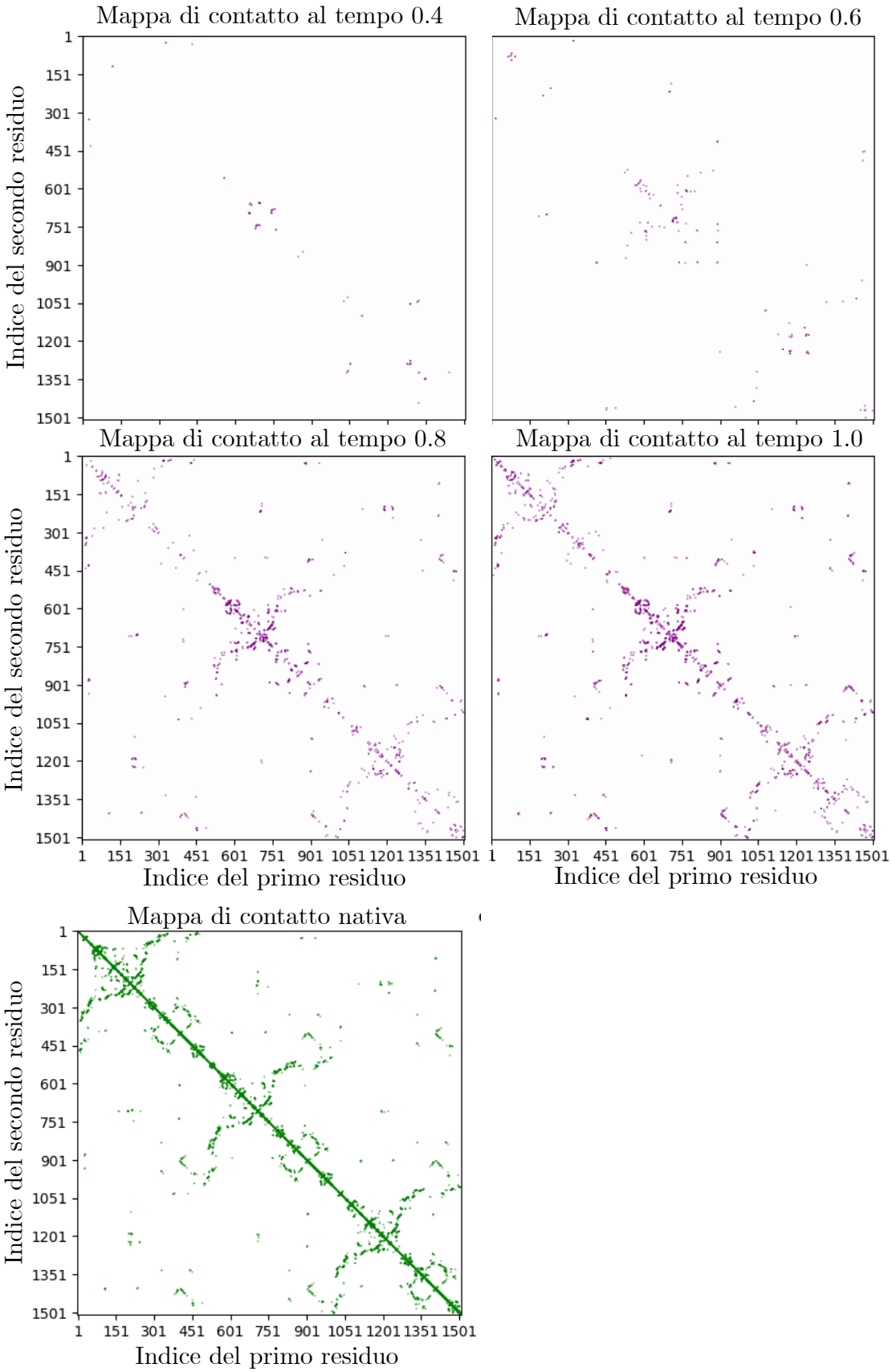


Figura 5.6: In viola le mappe di contatto della proteina 2HMG predette dall'algoritmo di ottimizzazione stocastica all'aumentare del tempo. In verde la mappa di contatto nativa

Capitolo 6

Conclusioni

6.1 Conclusioni

In conclusione in questo progetto di tesi è stato sviluppato un modello di Machine Learning che, combinato con un algoritmo di ottimizzazione stocastica, permette di studiare in dettaglio il processo di ripiegamento delle proteine permettendo di ottenere informazioni importanti sui contatti tra gli amminoacidi che sono protagonisti nel folding.

In particolare per la proteina 1PGB è stata sviluppata una rete neurale feed forward che associa ad una mappa di contatto il tempo normalizzato del processo di ripiegamento a cui tale mappa è associata. Il modello ottenuto ha una precisione sul training dataset dello 0.7% e sul test dataset del 5.5%, come mostrato in Fig. 5.1.

Dopo aver realizzato il modello, ne è stata testata l'accuratezza confrontando le distribuzioni dei tempi predetti coi tempi veri, che risultano essere entrambe uniformi, e studiando la correlazione tra tali tempi, che mostra un indice di correlazione di Pearson pari a $R = 0.99$. Quindi il modello è accurato e preciso nella predizione dei tempi normalizzati, come mostrato in Fig. 5.2.

Per verificare ulteriormente che il modello restituisca tempi fisicamente accettabili rispetto alla formazione dei contatti sulla mappa, è stata decomposta la struttura nativa della proteina 1PGB in 7 parti (Fig. 4.4) e sono state realizzate altrettante distribuzioni di mappe di contatto a cui è stato aggiunto un rumore gaussiano alla struttura nativa di base. Fornendo in input tali mappe alla FFNN, sono state ottenute delle distribuzioni di tempi che da 0 si spostano progressivamente verso l'1 man mano che si formano le strutture secondarie e terziarie della proteina. Quindi il modello restituisce risultati accettabili, come mostrato in Fig. 5.3.

In seguito, tramite un algoritmo di ottimizzazione, è stato possibile associare ad un tempo normalizzato, fornito come input, la mappa più probabile secondo

la rete neurale associata a tale tempo tramite un metodo interamente computazionale e applicabile a qualsiasi tipo di proteina.
Studiando la distribuzione dei contatti che l'algoritmo restituisce a tempi diversi inseriti in input, è stato possibile discretizzare le mappe predette ed ottenere delle mappe di contatti a valori binari e paragonabili con la mappa di contatto nativa. Le mappe di contatto ottenute mostrano che inizialmente si formano i contatti nativi della struttura secondaria tramite la formazione di un'α-elica e due β-forcine, in seguito si forma la struttura terziaria tramite l'unione tra loro delle strutture secondarie, come mostrato un Fig. 5.4. La mappa al tempo 1 presenta un grande numero di contatti nativi ed è molto simile alla struttura nativa effettiva della proteina 1PGB. Questo mostra che l'algoritmo di ottimizzazione combinato con il modello realizzato permette di indagare correttamente il processo di folding proteico, mostrando mappe di contatto relative a diversi istanti del processo di ripiegamento.

Realizzato per la proteina 1PGB, si è cercato di applicare lo stesso metodo ad una proteina estremamente più complessa, la 2HMG. Studiando la precisione del modello in funzione del numero di traiettorie impiegate nel dataset si è osservato che con un training dataset formato da una singola traiettoria di folding, l'errore in percentuale sul tempo predetto è del 14%, come illustrato in Fig. 4.6.
É stata quindi realizzata una sola simulazione di folding per la emoagglutinina, si è sviluppato un modello tramite una FFNN che, impiegato in un algoritmo di ottimizzazione, permette di ottenere anche in questo caso le mappe di contatto più probabili in funzione del tempo richiesto.
La FFNN ottenuta ha una precisione sul training dataset dell' 1.1% e le mappe di contatto ottenute mostrano i contatti tra gli amminoacidi che permettono di formare la struttura tridimensionale della proteina 2HMG, come mostrato in Fig. 5.6

Appendice A

Approfondimenti di Machine Learning

A.1 Funzione Costo, Retropropagazione e SGD

Introdotti i meccanismi e i concetti fondamentali su cui si basa il Machine Learning, in questa sezione si trattano rigorosamente i metodi matematici e gli algoritmi che vengono utilizzati in questa branca dell'informatica [37].
In particolar modo viene illustrato il metodo di apprendimento delle FFNN utilizzate in questo progetto di tesi.
Innanzitutto si comincia dalla definizione di un dataset.

$$D = \left\{ \big(\mathbf{X}, y\big)_i : y_i \in \mathbb{R}, X_i \in \mathbb{R}^{n-1}, i = 1, ..., p \right\}$$

Dove $\mathbf{X}_i^T = (X_{i,1}, ..., X_{i,n-1})$ e $\mathbf{y} = (y_1, ..., y_p)$.
Si vuole trovare il predittore f che descrive meglio il modello.
In un problema non lineare come può essere una FFNN i parametri e il predittore sono più complessi per questo si utilizza una funzione costo generale:

$$C(\{\theta\}) = \frac{1}{N} \sum_{i=1}^{N} \big[y_i - \hat{f}(x_i)\big]^2 = \frac{1}{N} \sum_{i=1}^{N} L\big(\mathbf{X}_i, \{\theta\}\big)$$

Dove $\{\theta\}$ è l'insieme dei parametri da minimizzare, L la funzione di perdita dopo l'iterazione di un singolo campionamento ed N è la dimensione del campionamento (batch-size).
Quando si addestra un modello di Machine Learning, solitamente non si utilizza l'intero dataset di addestramento in una volta sola, poiché ciò potrebbe richiedere troppa memoria e rallentare l'addestramento. Si preferisce suddividere il dataset in campionamenti (batch) più piccoli e si addestra il modello iterativamente su ciascuno di questi. La dimensione del campionamento (batch size) è il numero di esempi di addestramento in ciascun campionamento.
Adesso si deve trovare un metodo per minimizzare la funzione costo, ma con metodi deterministici si cade facilmente in un minimo locale da cui non si riesce

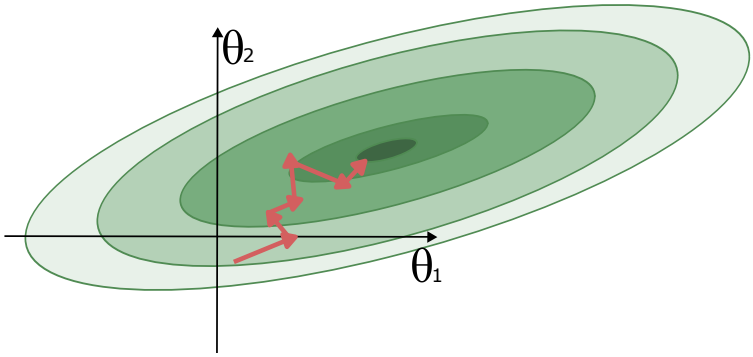


Figura A.1: Discesa Stocastica del Gradiente per un dataset con due parametri: θ_1 e θ_2

ad uscire. Per ovviare questo problema si utilizzano metodi stocastici come la *discesa stocastica del gradiente* (SGD) (Fig.A.1). L'idea è calcolare la funzione costo su un mini-batch contenenti $N_B \ll N$ elementi:

$$\boldsymbol{g}_{SGD}(\boldsymbol{\theta}) = \frac{1}{N_B}\sum_{i=1}^{N_B} \boldsymbol{\nabla}_\theta L(\mathbf{X}_i, y_i, \{\theta\})$$

Si usa un mini-batch perchéin questo modo il gradiente stocastico può far uscire da un minimo locale l'algoritmo di ricerca del minimo.
Si può mostrare che la SGD si può scrivere in una forma ottimale, come somma della funzione costo dipendente solo da θ e un rumore gaussiano $\mathcal{N}$:

$$\boldsymbol{g}_{SGD}(\boldsymbol{\theta}) = \frac{1}{N_B}\sum_{i=1}^{N_B} \vec{\nabla}_\theta C(\theta) + \mathcal{N}(0, \sigma^2)$$

Con questa riscrittura si ottiene che all'iterazione $\tau + 1$ i parametri θ assumono il valore:

$$\boldsymbol{\theta}_{\tau+1} = \boldsymbol{\theta}_\tau - \eta \nabla_\theta \boldsymbol{g}_{SGD}(\boldsymbol{\theta}_\tau) = \boldsymbol{\theta}_\tau - \eta \boldsymbol{\nabla}_\theta C(\boldsymbol{\theta}) + \eta \mathcal{N}(o, \sigma^2) \tag{A.1}$$

Dove η è il *learning rate* (tasso di apprendimento), un parametro che controlla l'ampiezza dei passi durante l'ottimizzazione dei parametri θ del modello nel processo di addestramento.
In sostanza, il learning rate determina quanto velocemente o lentamente un modello di Machine Learning convergerà verso i parametri ottimali. Un learning rate troppo piccolo può causare un addestramento molto lento e aumenta il rischio che l'algoritmo si arresti in presenza di minimi locali, mentre un learning rate troppo grande può causare oscillazioni intorno al minimo globale o addirittura divergere, come mostrato in Fig. A.2. L'equazione A.1 è molto simile ad una equazione di Langevin 1.5, infatti $T \propto \eta$ e $\langle \boldsymbol{\nabla} V \rangle \propto \frac{1}{N_B}$ dove T è la temperatura e $\langle \boldsymbol{\nabla} V \rangle \propto \frac{1}{N_B}$ la varianza del gradiente del potenziale. Quindi per essere più vicini al minimo si può diminuire η o aumentare il mini-batch N_B.
Si consideri un dataset più generale per una FFNN:

$$D = \left\{ (\mathbf{X}, \mathbf{y})_i : \mathbf{y}_i \in \mathbb{R}^{\shortparallel}, X_i \in \mathbb{R}^{n-1}, i = 1, ..., p \right\}$$

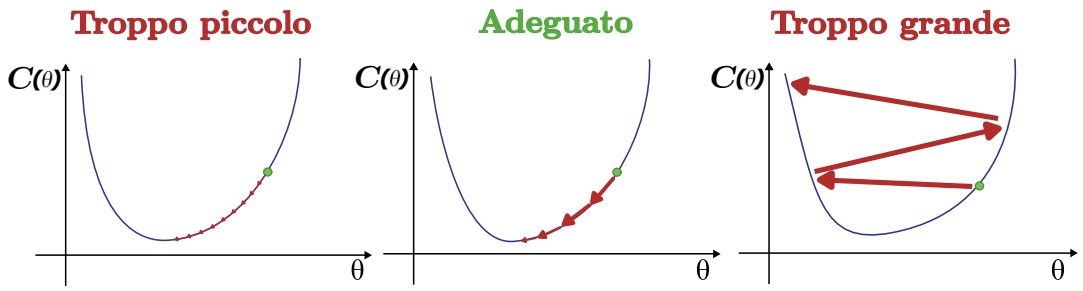


Figura A.2: Un piccolo tasso di apprendimento richiede tante iterazioni per raggiungere il minimo della funzione costo.
Un tasso di apprendimento ottimale descresce man mano che l'algoritmo si avvicina verso il minimo.
Un tasso di apprendimento troppo grande può portare a comportamenti divergenti.

In una FFNN il neurone artificiale 3.2 ha una componente lineare (somma pesata) e una componente non lineare (funzione di attivazione), quindi l'architettura di una rete generica è:

$$\begin{cases} z_j^{l=1} = \sum_{i=1}^{n(l)} w_{ij}^{l=1} x_i + b_j^{l=1} & \text{se } l = 1 \\ z_j^l = \sum_{i=1}^{n(l)} w_{ij}^l h_i^{l-1} + b_j^l & \text{se } 1 < l < L_{max} \\ \hat{y}_k = a(z_k^{L_{max}}) & \text{se } l = L_{max} \end{cases}$$

Dove z_j^l è l'input che arriva al neurone j dello strato l, $n(l)$ è il numero di neuroni nello strato l, w_{ij}^l è il peso del contatto tra il neurone i sullo strato $l-1$ e neurone j sullo strato l; b^l è il bias per lo strato l, L_{max} è il numero di strati totali, $h_i^{l-1} = f(z_i^l)$ dove f è la funzione di attivazione, $\hat{y}_k$ è il valore predetto dal modello con $k = 1, ..., K_{max}$ dove K_{max} è la dimensione dell'output, infine a è la funzione di attivazione dell'ultimo strato che può essere differente.
La funzione costo per una rete neurale con la struttura descritta precedentemente ha la seguente forma:

$$C(\{y_i\}, \{\mathbf{X_i}\}, \{\theta\}) = \frac{1}{N} \sum_{i=1}^{N} \tilde{C}(\mathbf{y_i}, \hat{\mathbf{y}}_\mathbf{i}, \theta) \tag{A.2}$$

Dove $\{\theta\} = \{w_{ij}^l, b^l, l = 1, ..., L_{max}\}$ è l'insieme dei parametri da ottimizzare per minimizzare la funzione costo, mentre $\tilde{C}$ è la funzione costo su un singolo campionamento (batch).
Per applicare la SGD alla FFNN ho bisogno delle derivate in tutte le direzioni, dopodiché, partendo dallo strato più esterno, si aggiornano i pesi e si procede verso l'interno. Da qui il nome *Retropropagazione* (Backpropagation).

A partire dall'ultimo strato $l = L_{max}$ minimizzo la funzione costo calcolando le derivate parziali rispetto ai parametri:

$$\frac{\partial \tilde{C}(\mathbf{y_i}, \hat{\mathbf{y}}_\mathbf{i}, \theta)}{\partial \theta_r^{L_{max}}} = \sum_{k=1}^{K_{max}} \frac{\partial \tilde{C}}{\partial z_k^{L_{max}}} \cdot \frac{\partial z_k^{L_{max}}}{\partial \theta_r^{L_{max}}} = \sum_{k=1}^{K_{max}} \delta_k^{L_{max}} \cdot \frac{\partial z_k^{L_{max}}}{\partial \theta_r^{L_{max}}}$$

dove si è definita come:

$$\sum_{K=1}^{K_{max}} \frac{\partial \tilde{C}}{\partial z_k^{L_{max}}} = \sum_{K=1}^{K_{max}} \frac{\partial \tilde{C}}{\partial \hat{y}_k} \cdot \frac{\partial \hat{y}_k}{\partial z_k^{max}} = \sum_{K=1}^{K_{max}} \frac{\partial \tilde{C}}{\partial \hat{y}_k} \cdot \frac{\partial a_k}{\partial z_k^{L_{max}}}\bigg|_{z_k^{max}} =: \delta_k^{L_{max}}$$

Sostituendo $z_k^L = \sum_{i=1}^{n(L_{max})} w_{ik}^{L_{max}} h_i^{L_{max}-1} + b_k^{L_{max}}$ si può scomporre $\frac{\partial z_k^{L_{max}}}{\partial \theta_r^{L_{max}}}$ e ottenere:

$$\frac{\partial z_K^L}{\partial w_{ij}^{L_{max}}} = \delta_k^{L_{max}} h_i^{L_{max}-1} \qquad \wedge \qquad \frac{\partial z_K^{L_{max}}}{\partial b_j^{L_{max}}} = \delta_k^{L_{max}}$$

Per cui si ottiene

$$\frac{\partial \tilde{C}(\vec{y_i}, \hat{\vec{y}_i}, \theta)}{\partial \theta_r^{L_{max}}} = \begin{cases} \dfrac{\partial \tilde{C}}{\partial w_{ij}^{L_{max}}} = \displaystyle\sum_{k=1}^{K_{max}} w_{ij} \delta_k^{L_{max}} \delta_{kj} h_i^{L_{max}-1} = \delta_j^{L_{max}} h_i^{L_{max}-1} \\ \dfrac{\partial \tilde{C}}{\partial b_j^{L_{max}}} = \displaystyle\sum_{k=1}^{K_{max}} \delta_k^{L_{max}} \delta_{kj} h_i^{L_{max}-1} = \delta_j^{L_{max}} \end{cases}$$

Così si possono aggiornare i pesi e il bias di conseguenza:

$$w_{ij} \leftarrow w_{ij} - \eta \delta_j^{L_m ax} h_i^{L_{max}-1} \qquad \wedge \qquad b_j^{L_{max}} \leftarrow b_j^{L_{max}} - \eta \delta_j^{L_{max}}$$

Per generalizzare ad uno strato che non sia l'ultimo si ha:

$$\frac{\partial \tilde{C}}{\partial \theta_r^l} = \sum_k \frac{\partial \tilde{C}}{\partial z_k^{L_{max}}} \cdot \frac{\partial z_k^{L_{max}}}{\partial \theta_k^l} = \sum_{k,i,j,\ldots,t} \delta_k^{L_{max}} \cdot \frac{\partial z_i^{L_{max}-1}}{\partial z_j^{L_{max}-2}} \cdot \ldots \cdot \frac{\partial z_t^{l+1}}{\partial z_s^l} \cdot \frac{\partial z_s^l}{\partial \theta_r^l}$$

Si nota che:

$$\frac{\partial z_t^{l+1}}{\partial z_s^l} = \sum_i \frac{\partial z_t^{l+1}}{\partial h_i^l} \frac{\partial h_i^l}{\partial z_s^l} = \sum_i w_{it} \delta_{is} \frac{\partial f}{\partial z}\bigg|_{z_s^l} = \sum_i w_{st} f'(z_s^l)$$

Per cui:

$$\delta_s^l = \sum_i \delta_t^{l+1} w_{st} f'(z_s^l)$$

Si è quindi ricavata una relazione ricorsiva per trovare δ_j^l:

$$\begin{cases} \delta_j^l = \displaystyle\sum_i \delta_i^{l+1} w_{ji} f'(z_j^l) & \text{se } l < L_{max} \\ \delta_k^{L_{max}} = \displaystyle\sum_k \frac{\partial a_k}{\partial z_k^{L_{max}}}\bigg|_{z_k^{max}} & \text{se } l = L_{max} \end{cases}$$

In questo modo si è in grado di calcolare analiticamente δ_j^l per ogni valore di $l = 1, ..., L_{max}$. Si comincia da $\delta_k^{L_{max}}$, poi $\delta_j^{L_{max}-1}$ e si itera il procedimento fino a δ_j^l.
Si è in grado di risolvere $\frac{\partial \tilde{C}}{\partial \theta_r^l} = ... = \sum_i \delta_s^l \frac{\partial z_s^l}{\partial \theta_r^l}$ con un calcolo analogo a quanto fatto in precedenza. In seguito si valuta $\frac{\partial z_s^l}{\partial \theta_r^l}$ e si aggiornano i pesi:

$$w_{ij}^l \leftarrow w_{ij}^l - \eta \delta_j^l h_i^{l-1} \qquad \wedge \qquad b_j^l \leftarrow b_j^l - \eta \delta_j^l$$

Tramite l'algoritmo di retropropagazione e la SGD si può allenare la rete neurale e aggiornare i pesi. Ripetendo per un numero adeguato di iterazioni sull'intero dataset (epoche) il modello aggiusta i pesi ogni volta per ottenere un output sempre più accurato.
Il numero di epoche è importante da selezionare durante la progettazione e l'addestramento del modello, infatti troppe poche epoche potrebbero non essere sufficienti per far convergere il modello, mentre troppe epoche potrebbero portare all'overfitting. L'*overfitting* si verifica quando un modello di Machine Learning si adatta troppo bene ai dati di addestramento e perde la capacità di generalizzare su nuovi dati non visti durante l'addestramento.
Dopo ogni epoca, è possibile valutare le prestazioni del modello su un set di dati di validazione (o test) per monitorare l'apprendimento e rilevare eventuali problemi.

Bibliografia

[1] T. E. Creighton, "Protein folding." *Biochemical journal*, vol. 270, no. 1, p. 1, 1990.

[2] H. H. Guo, J. Choe, and L. A. Loeb, "Protein tolerance to random amino acid change," *Proceedings of the National Academy of Sciences*, vol. 101, no. 25, pp. 9205–9210, 2004.

[3] F. Noé, G. De Fabritiis, and C. Clementi, "Machine learning for protein folding and dynamics," *Current opinion in structural biology*, vol. 60, pp. 77–84, 2020.

[4] C. I. Branden and J. Tooze, *Introduction to protein structure*. Garland Science, 2012.

[5] J. C. Whisstock and A. M. Lesk, "Prediction of protein function from protein sequence and structure," *Quarterly reviews of biophysics*, vol. 36, no. 3, pp. 307–340, 2003.

[6] "Immagine struttura primaria, secondaria e terziaria di wikipedia," https://upload.wikimedia.org/wikipedia/commons/c/c9/Main_protein_structure_levels_en.svg, accesso: 25/03/2024.

[7] S. Moore and W. H. Stein, "Chemical structures of pancreatic ribonuclease and deoxyribonuclease," *Science*, vol. 180, no. 4085, pp. 458–464, 1973.

[8] M. Sela, F. H. White Jr, and C. B. Anfinsen, "Reductive cleavage of disulfide bridges in ribonuclease," *Science*, vol. 125, no. 3250, pp. 691–692, 1957.

[9] C. B. Anfinsen, E. Haber, M. Sela, and F. White Jr, "The kinetics of formation of native ribonuclease during oxidation of the reduced polypeptide chain," *Proceedings of the National Academy of Sciences*, vol. 47, no. 9, pp. 1309–1314, 1961.

[10] C. B. Anfinsen and E. Haber, "Studies on the reduction and re-formation of protein disulfide bonds," *Journal of Biological Chemistry*, vol. 236, no. 5, pp. 1361–1363, 1961.

[11] H. W. Frederick Jr, J. Bello, D. Harker, and E. De Jarnette, "Regeneration of native secondary and tertiary structures by air oxidation of reduced ribonuclease," *Journal of Biological Chemistry*, vol. 236, no. 5, pp. 1353–1360, 1961.

[12] C. B. Anfinsen, "Principles that govern the folding of protein chains," *Science*, vol. 181, no. 4096, pp. 223–230, 1973.

[13] A. V. Finkelstein, A. Y. Badretdinov, and A. M. Gutin, "Why do protein architectures have boltzmann-like statistics?" *Proteins: Structure, Function, and Bioinformatics*, vol. 23, no. 2, pp. 142–150, 1995.

[14] S. S. Plotkin and J. N. Onuchic, "Understanding protein folding with energy landscape theory part i: basic concepts," *Quarterly reviews of biophysics*, vol. 35, no. 2, pp. 111–167, 2002.

[15] C. J. Guerriero and J. L. Brodsky, "The delicate balance between secreted protein folding and endoplasmic reticulum-associated degradation in human physiology," *Physiological reviews*, vol. 92, no. 2, pp. 537–576, 2012.

[16] P. J. Carter, "Introduction to current and future protein therapeutics: a protein engineering perspective," *Experimental cell research*, vol. 317, no. 9, pp. 1261–1269, 2011.

[17] S. Ishiwata, "On the on-line journal "biophysics and physicobiology (bppb)"," *Biophysical Reviews*, vol. 12, no. 2, pp. 217–219, 2020.

[18] J. K. Noel, M. Levi, M. Raghunathan, H. Lammert, R. L. Hayes, J. N. Onuchic, and P. C. Whitford, "Smog 2: a versatile software package for generating structure-based models," *PLoS computational biology*, vol. 12, no. 3, p. e1004794, 2016.

[19] M. Levi, P. Bandarkar, H. Yang, A. Wang, U. Mohanty, J. K. Noel, and P. C. Whitford, "Using smog 2 to simulate complex biomolecular assemblies," *Biomolecular Simulations: Methods and Protocols*, pp. 129–151, 2019.

[20] C. Clementi, H. Nymeyer, and J. N. Onuchic, "Topological and energetic factors: what determines the structural details of the transition state ensemble and "en-route" intermediates for protein folding? an investigation for small globular proteins," *Journal of molecular biology*, vol. 298, no. 5, pp. 937–953, 2000.

[21] J. K. Noel and J. N. Onuchic, "The many faces of structure-based potentials: from protein folding landscapes to structural characterization of complex biomolecules," in *Computational Modeling of Biological Systems: From Molecules to Pathways*. Springer, 2012, pp. 31–54.

[22] J. K. Noel, P. C. Whitford, K. Y. Sanbonmatsu, and J. . N. Onuchic, "Smog@ ctbp: simplified deployment of structure-based models in gromacs," *Nucleic acids research*, vol. 38, no. suppl_2, pp. W657–W661, 2010.

[23] P. C. Whitford, J. K. Noel, S. Gosavi, A. Schug, K. Y. Sanbonmatsu, and J. N. Onuchic, "An all-atom structure-based potential for proteins: bridging minimal models with all-atom empirical forcefields," *Proteins: Structure, Function, and Bioinformatics*, vol. 75, no. 2, pp. 430–441, 2009.

[24] J. K. Noel, P. C. Whitford, and J. N. Onuchic, "The shadow map: a general contact definition for capturing the dynamics of biomolecular folding and function," *The journal of physical chemistry B*, vol. 116, no. 29, pp. 8692–8702, 2012.

[25] H. Lammert, A. Schug, and J. N. Onuchic, "Robustness and generalization of structure-based models for protein folding and function," *Proteins: Structure, Function, and Bioinformatics*, vol. 77, no. 4, pp. 881–891, 2009.

[26] D. Van Der Spoel, E. Lindahl, B. Hess, G. Groenhof, A. E. Mark, and H. J. Berendsen, "Gromacs: fast, flexible, and free," *Journal of computational chemistry*, vol. 26, no. 16, pp. 1701–1718, 2005.

[27] M. Abraham *et al.*, "Gromacs 2024.1 manual," Feb. 2024. Available: https://doi.org/10.5281/zenodo.10721192

[28] H. Bekker, H. Berendsen, E. Dijkstra, S. Achterop, R. v. Vondrumen, D. Vanderspoel, A. Sijbers, H. Keegstra, and M. Renardus, "Gromacs-a parallel computer for molecular-dynamics simulations," in *4th international conference on computational physics (PC 92)*. World Scientific Publishing, 1993, pp. 252–256.

[29] H. J. Berendsen, D. van der Spoel, and R. van Drunen, "Gromacs: A message-passing parallel molecular dynamics implementation," *Computer physics communications*, vol. 91, no. 1-3, pp. 43–56, 1995.

[30] S. Pronk *et al.*, "Gromacs 4.5: a high-throughput and highly parallel open source molecular simulation toolkit," *Bioinformatics*, vol. 29, no. 7, pp. 845–854, 2013.

[31] E. Lindahl, B. Hess, and D. Van Der Spoel, "Gromacs 3.0: a package for molecular simulation and trajectory analysis," *Molecular modeling annual*, vol. 7, pp. 306–317, 2001.

[32] L. F. Shampine, "Stability of the leapfrog/midpoint method," *Applied mathematics and computation*, vol. 208, no. 1, pp. 293–298, 2009.

[33] P. Ongsulee, "Artificial intelligence, machine learning and deep learning," in *2017 15th international conference on ICT and knowledge engineering (ICT&KE)*. IEEE, 2017, pp. 1–6.

[34] T. Hastie, R. Tibshirani, J. H. Friedman, and J. H. Friedman, *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2009, vol. 2.

[35] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[36] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[37] K. P. Murphy, *Machine learning : a probabilistic perspective*. Cambridge, Mass. [u.a.]: MIT Press, 2013. Available: https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2

[38] N. Ketkar and J. Moolayil, *Introduction to PyTorch*. Berkeley, CA: Apress, 2021, pp. 27–91. Available: https://doi.org/10.1007/978-1-4842-5364-9_2

[39] W. Python, "Python," *Python Releases for Windows*, vol. 24, 2021.

[40] P. D. Bank, "Protein data bank," *Nature New Biol*, vol. 233, p. 223, 1971.

[41] CharlesHahn, "Charleshahn/duivytools: Schizoid man," Oct. 2023. Available: https://doi.org/10.5281/zenodo.8422182

[42] Q. Huang, R. P. Sivaramakrishna, K. Ludwig, T. Korte, C. Böttcher, and A. Herrmann, "Early steps of the conformational change of influenza virus hemagglutinin to a fusion active state: stability and energetics of the hemagglutinin," *Biochimica et Biophysica Acta (BBA)-Biomembranes*, vol. 1614, no. 1, pp. 3–13, 2003.

Ringraziamenti

Innanzitutto volevo ringraziare il mio professore, che mi ha introdotto e guidato come un vero maestro in questo percorso della fisica non scontandomi mai nulla, ma costruendo sempre.
I tesisti e i dottorandi con cui ho condiviso questo periodo di tesi che si è rivelato essere una grande esperienza di crescita personale.
Ringrazio di cuore tutti quelli che mi hanno accompagnato nella mia vita.
La mia famiglia per tutto quello che ha fatto per me affinché potessi crescere e sentirmi sempre voluto bene.
I miei amici di Bologna che mi hanno accompagnato durante elementari, medie e liceo e la cui amicizia continua ad essere un grande punto di compagnia e crescita anche ora.
I miei amici dell'università con cui è iniziata questa nuova avventura milanese, che mi accompagnano e mi aiutano nella quotidianità e coi alcuni dei quali sta fiorendo una amicizia per la vita.
Grazie a Enzo, a Giussani ed al Movimento che mi hanno mostrato che la vita è bella e vale la pena viverla anche dentro le fatiche.
Grazie alla mia morosa Eleonora.