
Università degli Studi di Milano

Facoltà di Scienze e Tecnologie
Corso di Laurea Triennale in Fisica

# Analisi di Traiettorie di Proteine con Metodi di Machine Learning

Relatore:
Prof. Guido Tiana .....

Candidato: Matteo Maria Bonamassa
Matricola: 848120
Codice P.A.C.S.: 87.15.-v

Sessione Laurea 24 aprile 2018
Anno Accademico 2016-2017

# Indice

# Introduzione

Simulare le proteine con metodi di dinamica molecolare è utile per capirne il funzionamento. Vi sono però due problemi fondamentali: uno è il peso computazionale delle simulazioni. Se voglio studiare la cinetica a temperatura fissata di una data proteina, non mi basterà una sola traiettoria ma dovrò ripetere tante simulazioni a partire da una stessa configurazione iniziale. Mentre, se voglio invece studiarne le proprietà all'equilibrio, dovrò fare una lunga simulazione che esplori eusastivamente lo spazio delle fasi.

Tuttavia, anche se risolvessimo questo problema, rimarrebbe comunque l'altra grande difficoltà, ovvero l'interpretazione dei risultati delle simulazioni. Le conformazioni delle proteine sono definite infatti in spazi di dimensione molto elevata. Riuscire a dare una descrizione consistente di una proteina in uno spazio di dimensionalità ridotta sarebbe molto utile. Basti pensare che se riuscissi a proiettare la dinamica di una proteina in 2 dimensioni, potrei disegnare la sua traiettoria con una curva in un grafico bidimensionale, cosa che riesce impossibile avendo 3N coordinate (tre per ognuno degli N atomi).

Cercare quindi di apporre una riduzione dimensionale al sistema è un obiettivo ragionevole, purtroppo non così semplice. Infatti non qualsiasi riduzione dimensionale va bene. Innanzitutto vogliamo che punti vicini nello spazio 3N-dimensionale siano anche vicini nello spazio ridotto. Per descrivere un sistema all'equilibrio, poi, è sufficiente che stati diversi (corrispondenti a minimi locali di energia) vengano mappati in stati differenti. Per studiarne la dinamica, oltre a questo, dobbiamo assicurarci che le coordinate ridotte rappresentino i moti più lenti del sistema, e che quindi la dinamica che uno può seguire lungo queste variabili sia la dinamica interessante del sistema [9].

Il lavoro di questa tesi è quindi focalizzato sull'implementazione e sull'utilizzo di reti neurali artificiali (algoritmi di apprendimento automatico per l'analisi dati) per lo studio di traiettorie di proteine estratte da simulazioni di dinamica molecolare, operando una riduzione dimensionale sulle coordinate del sistema.

Segue una piccola introduzione sulla teoria delle reti neurali artificiali. Viene poi descritto un primo lavoro di classificazione di traiettorie estratte da simulazioni di dinamica molecolare del dominio 1 della proteina G (che chiameremo proteina G per semplicità) e dell'HIV proteasi. Infine si presenta

in che modo si è quindi cercato di apporre una riduzione dimensionale ai sistemi studiati e che risultati si sono ottenuti.

# Capitolo 1

# Le proteine e i metodi tradizionali per descrivere il loro spazio conformazionale

Dato che il mio lavoro si è concentrato sullo studio di un metodo per estrarre variabili ridotte di sistemi all'equilibrio, vediamo quindi qualche intuizione in più riguardante le varibili ridotte e su come estarle.

## 1.1 Un esempio di variabile ridotta: l' RMSD

Con variabili ridotte si intende un certo insieme di variabili, funzioni delle coordinate originarie del sistema, in grado di apporre una riduzione dimensionale soddisfacente, ossia per cui stati diversi (corrispondenti a minimi locali di energia) vengano mappati in stati differenti. I parametri d'ordine sono particolari quantità che vengono usate per determinare quando il sistema si trovi in uno stato piuttosto che in un altro. Dipendono molto dal sistema che si decide di studiare: ad esempio, quando si vuole indagare il folding di una proteina, un parametro d'ordine molto utilizzato per distinguere gli stati foldati da quelli non foldati è l'RMSD, definita come:

$$RMSD(\{r_i\}) = \min_{rt} \left[ \frac{1}{N} \sum_{i=1}^{N} \left| r_i - r_i^{(nat)} \right| \right]^{\frac{1}{2}} \tag{1.1}$$

dove $min_{rt}$ sta a significare il minimo su tutte le possibili roto-traslazioni e $r_i^{(nat)}$ sono le coordinate corrispondenti della configurazione nativa. Tale variabile riesce quindi ad individuare i due stati più popolati corrispondenti ai minimi di energia libera del sistema. Tuttavia un problema rilevante è che il suo impiego ci fa perdere informazione sui grandi cambi conformazionali del sistema nello stato denaturato: infatti, dato che tali conformazioni sono già molto differenti da quella dello stato nativo, l'RMSD subirà solo lievi

cambiamenti. L' RMSD riflette quindi solo la parte più unfoldata della proteina data una singola conformazione, non fornendo informazione invece sui piccoli cambi conformazionali che possono avvenire lungo la catena.

Ma come si può fare per definire nuove variabili ridotte?

## 1.2 Principal Component Analysis

Per definire nuove variabili ridotte, si può partire da studi empirici fatti sul sistema da indagare, come nel caso dell'RMSD, oppure si può fare affidamento su strumenti più teorici che siano in grado di restituirmi una combinazione di variabili microscopiche che riproducano la termodinamica del sistema.

Una delle tecniche più efficaci di riduzione dimensionale è l'analisi a componenti principali (*Principal Component Analysis* - PCA). L'intuizione è la seguente: dato un insieme $S = \{\mathbf{x}_1, ..., \mathbf{x}_m\} \subseteq \mathbb{R}^d$ vogliamo trovare il sottospazio k-dimensionale che approssimi meglio $S$ secondo un certo errore. Da un punto di vista formale, data una base ortonormale $\mathbf{u}_1, ..., \mathbf{u}_k$ (la quale definisce quindi un sottospazio k-dim $\mathcal{V} \subseteq \mathbb{R}^d$), indichiamo con $U = \big[\mathbf{u}_1, ..., \mathbf{u}_k\big]$ la matrice con i vettori di base come colonne e indichiamo con $P = UU^T$ l'operatore lineare che proietta un vettore da $\mathbb{R}^d$ a $\mathcal{V}$.

L'errore di approssimazione è dato quindi da:

$$\sum_{t=1}^{m} \big|\big|\mathbf{x}_t - P\mathbf{x}_t\big|\big|^2 \tag{1.2}$$

L'eq. (1.2) misura quindi quanto ciascun punto $\mathbf{x}_t$ è distante dalla sua proiezione $P\mathbf{x}_t$ nel sottospazio k-dim.

Ciò che fa la PCA è trovare la P nello spazio $\mathcal{P}_k$ di tutte le matrici di proiezione su sottospazi k-dim per cui (1.2) è minimo, ossia:

$$\min_{P \epsilon \mathcal{P}_k} \sum_{t=1}^{m} \big|\big|\mathbf{x}_t - P\mathbf{x}_t\big|\big|^2 \tag{1.3}$$

Possiamo ora riscrivere il problema risolto dalla PCA utilizzando la norma di Frobenius: data una matrice $V = \big[\mathbf{v}_1, ..., \mathbf{v}_m\big]$ con $\mathbf{v}_i \epsilon \mathbb{R}^d$, vale

$$||V||_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{d} V_{i,j}^2 = \sum_{i=1}^{m} ||\mathbf{v}_i||^2 \tag{1.4}$$

Se prendiamo ora $X$ come matrice le cui colonne sono gli elementi si $S$, allora

$$\sum_{t=1}^{m} \big|\big|\mathbf{x}_t - P\mathbf{x}_t\big|\big|^2 = ||X - PX||_F^2 \tag{1.5}$$

e quindi il problema si riduce a

$$\min_{P \epsilon \mathcal{P}_k} ||X - PX||_F^2. \tag{1.6}$$

Per risolvere questo problema possiamo utilizzare il concetto di decomposizione sui valori singolari ( *Singular Value Decomposition* - SVD) di una matrice. Data infatti una matrice $X$ reale $d \times m$, esistono una matrice diagonale $(d \times m)$ $\Sigma = diag(\sigma_1, ..., \sigma_r)$ con diagonale $\sigma_1 \geqslant ... \geqslant \sigma_r \geqslant 0$ $(r = min\{d, m\}$ ) e due matrici $U$ $(d \times d)$ e $V$ $(m \times m)$ tali che $X = U\Sigma V^T$. Gli scalari $\sigma_1, ..., \sigma_d$ sono detti *valori singolari* di $X$. Inoltre le matrici $U$ e $V$ sono ortogonali.

Si può dimostrare che la migliore approssimazione di una matrice $X$ fra tutte le matrici di rango al più k è la matrice ottenuta ponendo a zero i valori singolari più piccoli di $\sigma_k$.

Ritornando quindi alla PCA si ha che P, proiettore sullo spazio k-dim, deve avere ovviamente rango al più k. Quindi cerchiamo P di rango al pi k tale che $PX = X_k$. Scegliendo $P = UU^T$, dove $U_k = [\mathbf{u}_1, ..., \mathbf{u}_k, \mathbf{0}, ..., \mathbf{0}]$ è la matrice le cui prime k componenti coincidono con quelle delle matrice U della SVD di $X = U\Sigma V^T$, otteniamo:

$$PX = U_k U_k^T X = U_k U_k^T U \Sigma V^T = U_k \Sigma V^T = U \Sigma_k V^T \tag{1.7}$$

dove l'ultima uguaglianza vale perchè le colonne di $U_k$ dopo le prime k sono tutte $\mathbf{0}$. Riassumendo, tramite la PCA rappresentiamo quindi ciascun punto $\mathbf{x} = (x_1, ..., x_d)$ tramite le k features $\mathbf{x}' = U_k \mathbf{x}$ che sono la migliore approssimazione k-dim del punto $\mathbf{x}$.

Uno dei limiti più grandi della PCA sta però nel fatto che in nessun modo ci viene garantito che la dimensione k sia molto più piccola di d, cosa che ho riscontrato anche io durante il mio lavoro.

# Capitolo 2

# Metodi di apprendimento automatico

Il campo dell'apprendimento automatico (*machine learning*) è un ambito di studio molto vasto e diversificato. Le reti neurali artificiali sono un particolare esempio di apprendimento automatico supervisionato. Tali strumenti sono modelli computazionali ispirati alla struttura delle rete neurali cerebrali (da cui il nome): possiamo infatti vedere il cervello secondo un modello molto semplificato come insieme di un grande numero di unità computazionali di base (*neuroni*) collegate tra loro in complessi network comunicanti. Le reti neurali artificiali sono quindi costruiti in analogia con tale modello. Sono utilizzate nei più diversi campi per risolvere differenti tipi di problemi di inferenza ad esempio: categorizzazione e partizionamento di dati, predizioni di serie temporali, pianificazione di sequenze di azioni.

Si introduce quindi brevemente la teoria e la terminologia a sostegno delle reti neurali artificiali *feedforward* [11]. Segue una discussione sulle modalità di raccolta dati per l'addestramento.

## 2.1 Reti neurali artificiali feedforward

Possiamo descrivere una rete neurale artificiale *feedforward* attraverso un grafo diretto aciclico $G = (V, E)$, dove con grafo G = (V, E) si intende un insieme di elementi $V$, detti nodi, che possono essere collegati fra loro da linee, dette archi, appartenenti all'insieme $E$. Un grafo si di dice diretto se gli archi hanno un orientazione. Un grafo diretto può essere aciclico se comunque si scelga il vertice di partenza non possiamo tornare ad esso percorrendo gli archi del grafo (vedi Fig. 2.1).

Per specificare una data rete neurale bisognerà inoltre fornire una mappa $w : E \to \mathbb{R}$, altrimenti rappresentabile da una matrice $W$, che associ ad ogni arco $(i, j) \in E$ un parametro (detto peso) $w_{i,j} \in \mathbb{R}$. Ogni nodo è associato ad una funzione scalare $\sigma : \mathbb{R} \to \mathbb{R}$ solitamente scelta tra:

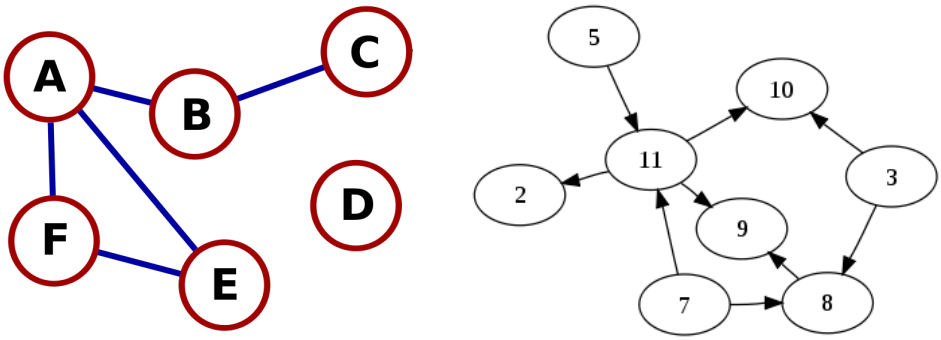



Figura 2.1: Sulla sinistra un Grafo generico con 6 nodi e 5 connessioni; sulla detsra un grafo connesso diretto aciclico. Si noti che si può decidere di etichettare i diversi nodi a propria discrezione.

- Funzione segno. $\sigma(a) = sign(a)$
- Funzione soglia. $\sigma(a) = \begin{cases} 1 & \text{se } a > 0 \\ 0 & \text{altrimenti} \end{cases}$
- Funzione sigmoide. $\sigma(a) = \dfrac{1}{1 + exp(-a)}$

Chiamiamo la funzione $\sigma$ "funzione di attivazione" del nodo. Definiamo invece "attivazione" del nodo il valore $a$ su cui viene calcolata $\sigma$.

È possibile ordinare il grafo in strati (*layer*) differenti all'interno dei quali si hanno diversi nodi (o neuroni) (vedi Fig. 2.2). Dividiamo quindi l'insieme di tutti i nodi in sottoinsiemi disgiunti (nonvuoti), $V = \bigcup_{t=0...T} V_t$ (dove $T$ indica il numero di layer detto profondità (*depth*) della rete), in cui ogni connessione in $E$ connette un nodo in $V_{t-1}$ a uno in $V_t$ per ogni $t \in [1, ..., T]$; se indichiamo con $W_t$ la matrice dei pesi tra il layer t e t+1, si avranno quindi $T$ matrici di dimensioni $|V_t| \times |V_{t+1}|$ (dove $|V_t|$ indica la dimensione (numero di nodi) del t-esimo layer) tali che $W = \bigcup_{t=1...T} W_t$. Il primo layer ($V_0$) è comunemente detto *input layer*. Esso contiene $n+1$ nodi (vedremo più avanti perchè "+1"), dove $n$ è la dimensione dello spazio dei dati di input. Denotiamo quindi con $v_{t,i}$ l'i-esimo nodo del t-esimo layer e con $o_{t,i}(\mathbf{x})$ l'output di $v_{t,i}$ (ossia la funzione di attivazione specifica del nodo calcolata nella sua attivazione) qualora il network sia fornito di un vettore di input $\mathbf{x}$. Si noti che dato $\mathbf{x} = \{x_1, ..., x_n\}$ per ogni $i \in [1, ..., n]$, l'output dell'i-esimo neurone in $V_0$ è semplicemente l'input $x_i$.

Supponendo di aver calcolato l'output dei neuroni nel layer $t$, possiamo calcolare gli output dei neuroni nel layer $t+1$. Fissato quindi $v_{t+1,j} \in V_{t+1,j}$ si ha:

$$o_{t+1,j}(\mathbf{x}) = \sigma(a_{t+1,j}(\mathbf{x})) \tag{2.1}$$

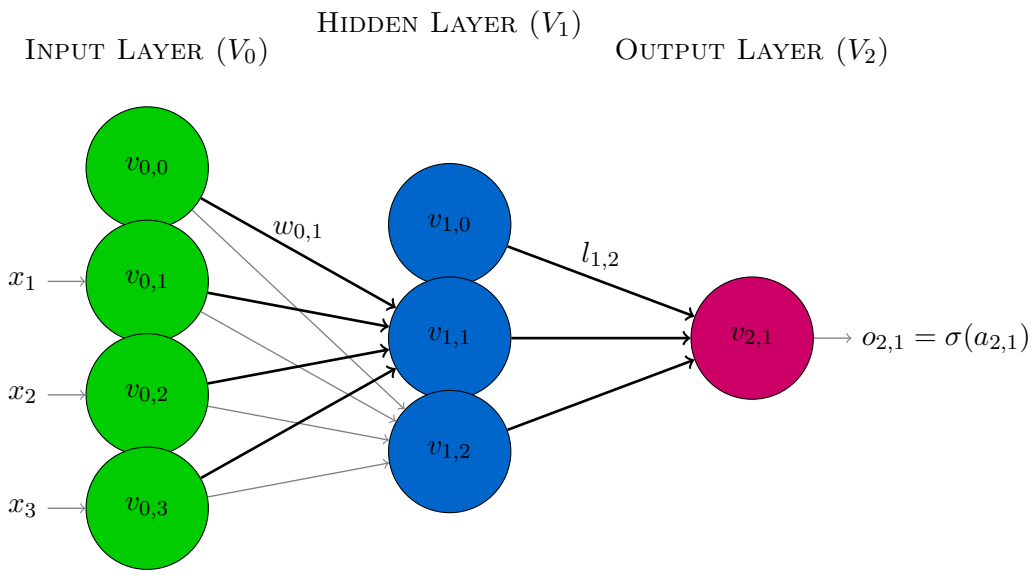



Figura 2.2: Esempio di rete neurale artificiale come grafo orientato in cui si ha un solo layer nascosto ($V_1$), un singolo output (rappresentato dalla funzione di attivazione di $v_{2,1}$ calcolata nella sua attivazione, $\sigma(a_{2,1})$) e un vettore di input $\mathbf{x} = \{x_1, x_2, x_3\}$. Si noti che $w_{1,2} \in W_1$ e $l_{1,2} \in W_2$. Sia nell'input che nell'hidden si noti la presenza di un nodo non collegato col resto della rete detto *bias*. Tale nodo ha sempre output 1 ed è inserito per motivi di convergenza per reti che presentano funzioni sigmoidi come funzioni di attivazione.

con,

$$a_{t+1,j}(\mathbf{x}) = \sum_{r:(v_{t,r},v_{t+1,j})\in E} w(v_{t,r}, v_{t+1,j}) o_{t,r}(\mathbf{x}) \tag{2.2}$$

In parole, l'attivazione di $v_{t+1,j}$ è quindi una somma pesata secondo $W$ di tutti gli output dei neuroni in $V_t$ che sono ad esso connessi e il suo output è quindi la funzione di attivazione applicata in $a_{t+1}$.

In ultimo è importante osservare che è ormai usanza comune inserire nei diversi layer (meno quello di output) un nodo aggiuntivo, detto nodo di *bias*, connesso con tutti i nodi del layer successivo ma con nessuno di quelli del layer precedente (si veda ancora la Fig. 2.2). Tale nodo ha sempre output uguale ad 1 e permette alla rete di traslare la funzione di attivazione all'occorrenza . L'aggiunta di tale parametro è quindi necessario per la convergenza della rete o per lo meno per renderla più veloce.

## 2.2 Apprendimento della rete

Una volta che si è specificata la propria rete tramite ( *V, E, σ, w*), si può pensare alla stessa come ad una funzione

$$\mathbf{h}_{V,E,\sigma,w} : \mathbb{R}^{|V_0|} \to \mathbb{R}^{|V_T|} \tag{2.3}$$

che in teoria dell'apprendimento automatico è conosciuta come funzione ipotesi o predittore. Parliamo quindi indistintamente di predittori e reti neurali. Si noti che tuttavia il concetto di predittore è un concetto più generale: esistono infatti predittori che non sono per forza reti neurali, si pensi ad esempio a predittori ad albero o a predittori a k-vicini. Si tratta di algoritmi usati con gli stessi scopi delle reti neurali ma che non sono stati studiati durante questo lavoro.

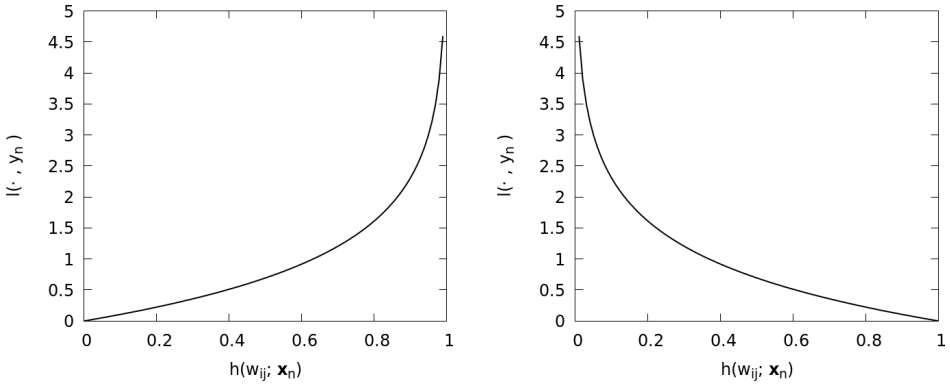
Possiamo quindi determinare delle classi di predittori fissando il grafo ($G = (V, E)$) e la funzione di attivazione $\sigma$, mentre facciamo variare $w : E \to \mathbb{R}$ (ossia la nostra matrice dei pesi $W$). La tripletta ( *V, E, σ*) è detta *architettura* della rete. Definiamo quindi una classe di ipotesi come:

$$\mathcal{H}_{V,E,\sigma} = \{\mathbf{h}_{V,E,\sigma,w} | \text{ w mappa da E a } \mathbb{R}, \text{ variabile}\}. \tag{2.4}$$

La matrice dei pesi $W$ sarà quindi ciò che distingue un predittore in una data classe di ipotesi. Saranno quindi tali pesi a determinare la bontà di un predittore. Vediamo meglio cosa si intende.

### 2.2.1 Algoritmi di apprendimento

Per capire quando una rete è un buon predittore o meno abbiamo bisogno di introdurre quella che in letteratura è conosciuta come funzione di rischio (o costo o errore) che misuri la discrepanza tra l'output predetto e quello




(a) $y_n = 0$ (b) $y_n = 1$

Figura 2.3: Funzio costo definita dall'eq. (2.5) spezzata nei due casi $y_n = 0, 1$ e per un singolo esempio $n$.

vero dato un set di esempi $\mathcal{S}_n = (\mathbf{x}_n, y_n)$ detto *training* set. Si noti che implicitamente si assume che il set di esempi derivi da una distribuzione $\mathcal{D}$ vera a noi sconosciuta; quello che abbiamo noi è solo un sample $\mathcal{S}_n$ di tale distribuzione.

Alcuni esempi di funzioni costo possono essere:

$$l_{\mathcal{S}_n}(h(w_{ij};\mathbf{x}_n), y_n) = -\frac{1}{N}\sum_{n=1}^{N}\left[y_n log(h(w_{ij};\mathbf{x}_n)) + (1-y_n) log(1-h(w_{ij};\mathbf{x}_n))\right] \tag{2.5}$$

$$l_{\mathcal{S}_n}(h(w_{ij};\mathbf{x}_n), y_n) = MSE(h(w_{ij};\mathbf{x}_n), y_n) \equiv \frac{1}{2N}\sum_{n=0}^{N}(h(w_{ij};\mathbf{x}_n) - y_n)^2 \tag{2.6}$$

dove la prima è una regressione logistica che viene solitamente usata per problemi di classificazione binaria (in cui cioè, mi aspetto che l'output $h(w_{ij};\mathbf{x}_n) \in \{0, 1\}$ o comunque molto prossimo a questi valori). Per come è costruita la funzione infatti, utilizzando l'eq. (2.5), si penalizzano molto (con valori grandi di $l(\cdot, y_n)$) le predizioni sbagliate e per niente quelle giuste (si veda la (Fig. 2.3) per una maggiore intuizione riguardo tale funzione costo). In questo modo si permette alla funzione costo di oscillare poco. La seconda equazione invece è generalmente usata per problemi di regressione in cui l'output è un valore reale compreso in qualche intervallo, per cui $l(\cdot, y_n)$ crescerà al crescere della distanza tra $h(\mathbf{x}_n)$ e $y_n$.

Mi aspetto quindi che minore sarà il valore della funzine costo, migliore sarà il mio predittore. Quello che vogliamo fare quindi è muoverci lungo la funzione costo in modo tale da raggiungerne il minimo; per fare ciò si

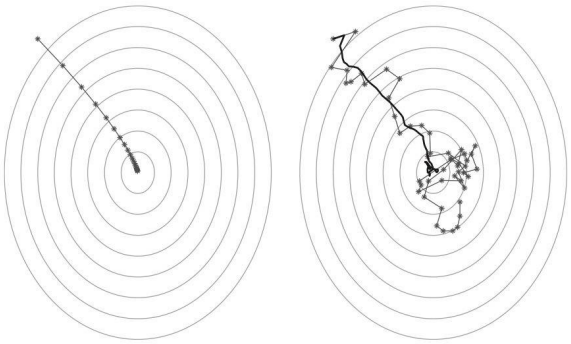


Figura 2.4: Illustrazione di un algoritmo batch (sulla sinistra) a di uno mini-batch stocastico (sulla destra). Si plottano quindi le curve di livello di una funzione di costo quadratica tridimensionale. I punti indicano i passi aggiornati secondo l'eq. (2.7) dell'algoritmo di apprendimento. Nel caso di destra, la linea solida rappresenta il valore medio del gradiente. [10]

hanno a disposizione diversi algoritmi iterativi. Questa fase, ossia quella in cui ricerco i pesi migliori per la mia rete è detta fase di *apprendimento* o *addestramento*. Si tenga conto sempre del fatto che non potremo mai minimizzare la vera funzione $l_{\mathcal{D}}(h)$costo dato che dipenderà dalla distribuzione $D$ a noi sconosciuta, ma minimizzeremo invece una funzione di costo empirica $l_{\mathcal{S}}(h)$ che mi aspetto essere almeno la maggiorante di $l_{\mathcal{D}}(h)$ (vedremo più avanti un metodo per dare una stima di $l_{\mathcal{D}}(h)$, detta funzione di costo statistica).

Le reti neuronali vengono tipicamente addestrate usando algoritmi che si basano sulla discesa del gradiente, in cui

$$w'_{ij} = w_{ij} - \eta \sum_{n} \frac{\partial l(h(w_{ij}; \mathbf{x}_n), y_n)}{\partial w_{ij}} \tag{2.7}$$

dove $\eta$ è il parametro conosciuto in letteratura come rate di apprendimento (*learning rate*), $l(\cdot, y_n)$ è una funzione costo scelta convessa ( come le due di esempio riportate sopra) e $w'_{ij}$ è l' ij-esimo peso aggiornato tale per cui $(i, j) \in E$. L'Eq. (2.7) ci fornisce quindi la regola iterativa per aggiornare i pesi ad ogni passo. La direzione verso cui ci si sposta ad ogni passo sarà quindi quella di massima variazione verso il minimo (si guardi la (Fig. 2.4) per maggiore intuizione). La variazione dei pesi sarà quindi più grande quanto più grande è $\eta$.

Una prima grande differenza importante tra i diversi algoritmi di apprendimento esistenti, è la cardinalità del sottoinsieme di esempi su cui si decide di sommare per il calcolo della derivata parziale: esistono infatti algoritmi in cui si prende un solo esempio estratto a caso alla volta detti algoritmi *online*; altri in cui si decide di prendere come sottoinsieme l'intero insieme di esem-

pi detti algoritmi *batch* e altri ancora in cui se ne prende un sottoinsieme casuale finito detti *mini-batched*. Questi ultimi algoritmi sono conosciuti anche come algoritmi di discesa del gradiente stocastico ( *Stochastic Gradient Descent* - SGD): seppur il sottoinsieme sia casuale, richiediamo comunque che il valore medio del gradiente calcolato sul sottoinsieme sia la direzione vera del gradiente. Guardando la (Fig. 2.4) si può avere un intuizione della differenza appunto tra tra algoritmi *batch* e *mini-batch*.

Per modificare i parametri dei layer intermedi durante la discesa del gradiente è necessario muoversi non solo in un senso, calcolando quindi quanto la predizione si discosti dal dato vero e aggiustare i pesi di conseguenza, ma anche nell'altro, propagando l'errore commesso all'indietro per aggiustare i pesi in base alla differenza tra input veri e decodificati. Tale algoritmo prende il nome di retropropagazione dell'errore. Vediamolo meglio.

Per comodità, supponiamo di considerare una rete con due strati nascosti, il primo con $l$ nodi e il secondo con $i$ nodi, mentre un solo nodo in output (lo stesso ragionamento è generalizzabile a casi con più strati nascosti e più nodi in output). Si ha inoltre un training set $\mathcal{S} = \{\mathbf{x}_n, y_n\}$. Per alleggerire la notazione, poniamo che 0 sia l'indice dell'unico nodo di output, quindi $a_0$ la sua attivazione. Si avrà $h_{G,W,\sigma}(\mathbf{x}_n) = \sigma(a_0) = v_0$. Per un qualunque nodo $i$ nel secondo strato nascosto, tale che $(i, 0) \in E$ (dove $(i, 0)$ indica l'arco tra il nodo i-esimo del secondo strato nascosto e l'unico nodo dell'output), applicando due volte la *chain-rule* alla derivata parziale per il calcolo della funzione costo per un n-esimo esempio, otteniamo

$$\frac{\partial l_n(W)}{\partial w_{i0}} = \frac{\partial l(v_0, y_n)}{\partial w_{i0}} = \frac{\partial l(v_0, y_n)}{\partial v_0}\frac{\partial v_0}{\partial a_0}\frac{\partial a_0}{\partial w_{i0}} = l'_{v_0}\frac{d\sigma(a_0)}{da_0}\frac{\partial a_0}{\partial w_{i0}}. \tag{2.8}$$

Le derivate $l'_{v_0}$ e $\sigma'(a_0)$, sono calcolabili direttamente a partire dalle funzioni di perdita e di attivazione scelte. Si noti quindi che entrambe devono essere scelte come differenziabili. Ricordando inoltre che, in questo caso, $a_0 = \mathbf{w}(0)^T\mathbf{v}(0)$ (con $\mathbf{w}(0)^T$ vettore dei pesi trasposto delle connessioni tra i nodi del secondo hidden layer e nodo in output e $\mathbf{v}(0)$ vettore degli output dei nodi connessi al nodo in output) abbiamo quindi

$$\frac{\partial a_0}{\partial w_{i0}} = v_i = \sigma(\mathbf{w}(i)^T\mathbf{v}(i)). \tag{2.9}$$

Possiamo quindi scrivere

$$\frac{\partial l_n(W)}{\partial w_{i0}} = l'_{v_0}\sigma'(a_0)v_i. \tag{2.10}$$

Calcoliamo ora $\frac{\partial l_n(W)}{\partial w_{li}}$ per i nodi l nel primo strato nascosto. Si noti che

$$
\begin{aligned}
v_0 = \sigma(\mathbf{w}(0)^T \mathbf{v}(0)) &= \sigma\big( \sum_{i:(i,0)\in E} w_{i,0}\sigma(\mathbf{w}(i)^T \mathbf{v}(i))\big) \\
&= \sigma\big( \sum_{i:(i,0)\in E} w_{i,0}\sigma\big( \sum_{l:(l,i)\in E} w_{l,i} v_l\big)\big).
\end{aligned}
\tag{2.11}
$$

Quindi, per ogni nodo l nel primo strato nascosto, denotando $a_l = \mathbf{w}(l)^T \mathbf{v}(l)$ possiamo scrivere

$$
\frac{\partial l_n(W)}{\partial w_{li}} = \frac{\partial l(v_0, y_n)}{\partial a_0}\frac{\partial a_0}{\partial v_i}\frac{\partial v_i}{\partial a_i}\frac{\partial a_i}{\partial w_{l,i}} = l'(v_0)\sigma'(a_0) w_{i,0}\sigma'(a_i) v_l. \tag{2.12}
$$

Possiamo riscrivere quanto fatto in modo più compatto sfruttando il fatto che per ogni nodo l

$$
\frac{\partial l_n(W)}{\partial a_l} = \sum_{i:(l,i)\in E} \frac{\partial l_n(W)}{\partial a_i}\frac{\partial a_i}{\partial v_l}\frac{\partial v_l}{\partial a_l} = \sum_{i:(l,i)\in E} \frac{\partial l_n(W)}{\partial a_i} w_{l,i}\sigma'(a_l) \tag{2.13}
$$

Quindi, introduccendo la definizione ricorsiva

$$
\delta_l = \frac{\partial l_n(W)}{\partial a_l} = \begin{cases} l'(v_0)\sigma'(a_0) & \text{se i = 0} \\ \sigma'(a_l)\sum_{i:(l,i)\in E} \delta_i w_{l,i} & \text{altrimenti} \end{cases} \tag{2.14}
$$

possiamo scrivere la derivata parziale per un qualunque $(i,j) \in E$ come

$$
\frac{\partial l_n(W)}{\partial w_{i,j}} = \frac{\partial l_n(W)}{\partial a_j}\frac{\partial a_j}{\partial w_{i,j}} = \delta_j v_i. \tag{2.15}
$$

Per avere una maggiore intuizione vedere lo pseudo codice (2.1)

Algorithm 2.1: Backpropagation

```
1  input: esempi (x_n, y_n), matrice dei pesi W, grafo (V,E), σ
2
3  inizializzazione: denotiamo gli strati del grafo V_0, ..., V_T
4  dove V_t = {v_{t,1}, ..., v_{t,k_t}} e definiamo a random
5  le t matrici dei pesi W_{t,i,j} tra i nodi (v_{t,j}, v_{t+1,i})
6
7  forward:
8  set o_0 = x
9  for t in {1..T}
10   for i in {1..k_t}
11     set a_{t,i} = Σ_{j=1}^{k_{t-1}} W_{t-1,j} o_{t-1,j}
12     set o_{t,i} = σ(a_{t,i})
13
14 backward:
15 set δ_t = o_T − y
16 for t in {T−1..1}
```

```
    for i in {1..k_t}
      set δ_{t,i} = Σ_{j=1}^{k_{t+1}} W_{t,j,i} δ_{t+1,j} o'_{a_{t+1,j}}

  output:
  per ogni arco (v_{t,j}, v_{t+1,i}) ∈ E
    set ∂l_n(W) / ∂w_{(v_{t,j}, v_{t+1,i})} = δ_{t,i} σ'(a_{t,i}) o_{t-1,j}
```

Addestrare una rete usando l'algoritmo di retropropagazione dell'errore non è tuttavia immediato; non è detto infatti che una volta implementato l'algoritmo e settato dei valori dei vari parametri (come $\eta$) che sembrano funzionare per la maggior parte dei problemi affrontati in letteratura, esso converga ad una soluzione che vada bene per il problema studiato. Per rendere possibile e facilitare una convergenza rapida ad un buon minimo locale, negli anni si sono studiate un insieme di euristiche come ad esempio: l'inizializzazione casuale dei pesi della rete in un certo intervallo di valori [7], l'inserimento di parametri di regolarizzazione all'interno della regola di aggiornamento dei pesi che non permettano a questi di divergere. Tuttavia, non essendoci ancora una scienza esatta dietro questi algoritmi di apprendimento automatico, anche con l'utilizzo di queste e altre euristiche sta all'utente, attraverso l'esperienza, capire cosa vada bene o meno per il problema indagato.

Si applica quindi l'algoritmo di apprendimento esplorando tutto lo spazio degli esempi a disposizione per diverse volte fino a convergenza. Il numero di volte che esploro tutto lo spazio degli esempi a disposizione coincide con il numero di ciò che viene chiamato "epoca".

Durante il mio lavoro ho quindi scelto e utilizzato principalmente due algoritmi conosciuti in letteratura come *Resilient Backpropagation* (RPROP) [8] e *Stochastic Gradient Descent* (SGD) [10]. Entrambi si basano su metodi di retropropagazione dell'errore, ma hanno alcune differenze fondamentali: (RPROP) fu uno dei primi algoritmi batch a mostrare velocità di convergenza apprezzabili grazie alla adattività di $\eta$ in base al segno della derivata della funzione di perdita, e all'inizializzazione random dei pesi in intervalli di valori studiati su gli input utilizzati. (SGD) tutt'oggi è uno degli algoritmi più utilizzati data la sua velocità di convergenza superiore addirittura di due ordini di grandezza come numero di epoche rispetto a RPROP come da me riscontrato. Tale velocità è imputabile alla sua natura stocastica per il calcolo della direzione del gradiente, che lo rende computazionalmente molto più leggero dato si avranno meno calcoli da fare (come specificato sopra ), e all'impiego di un parametro aggiuntivo nel calcolo dei nuovi pesi chiamato momento:

$$\Delta w_{ij}(t) = -\eta \frac{\partial l(W)}{\partial w_{ij}}(t) + \mu \Delta w_{ij}(t-1). \tag{2.16}$$

dove $\mu$ è il momento e $\Delta w_{ij}(t-1)$ è la differenza tra i pesi nuovi e vecchi all'epoca t-1. Tale parametro previene problemi dovuti ad un impostazione

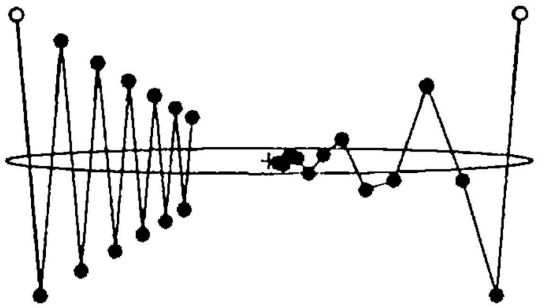


Figura 2.5: Discesa del gradiente su una funzione quadratica. Entrambe le traiettorie sono di 12 passi con uno stesso $\eta$. Sulla sinistra non si ha momento mentre sulla destra sì. [11]

del learning rate troppo basso, che determina piccoli passi lungo la funzione costo e quindi l'impiego di più epoche per il raggiungimento di una soluzione accettabile, o troppo alto, il quale al contrario non permetterebbe il raggiungimento di una data soglia e causerebbe una forte oscillazione se non una divergenza della soluzione (Fig.2.5).

È importante osservare che la funzione $l_{\mathcal{S}_n}(h(w_{ij}; \mathbf{x}_n), y_n)$ totale è non convessa anche quando scegliamo $l(\cdot, y_n)$ convessa per ogni $y_n$. Ciò ha come conseguenza importante che col metodo della discesa del gradiente spesso la funzione di rischio rimane intrappolata in un minimo locale.

In pratica però si osserva che i minimi locali del training error corrispondono in generale a predittori a basso rischio. Questo potrebbe significare che molti dei minimi locali sono vicini al minimo globale, non rendendo quindi necessario raggiungere un minimo assoluto della funzione di rischio per ottenere buone capacità di generalizzazione. Si intuisce che per la scelta del predittore non ci si possa affidare solamente al valore della funzione rischio sul training set. Vediamo quindi un altro metodo di validazione di un particolare predittore.

### 2.2.2 Validazione del predittore

Il calcolo di una eventuale funzione di errore o di altri parametri, come ad esempio il numero di esempi riprodotti correttamente, sul solo dataset usato durante l'apprendimento non è sufficiente a garantire la capacità predittiva di una rete. Essa infatti potrebbe aggiustare i propri pesi fino ad arrivare a predire perfettamente ogni esempio del training set pur tuttavia non riuscendo a generalizzare e quindi predire esempi al di fuori di questo set. Questo fenomeno prende il nome di *overfitting*.

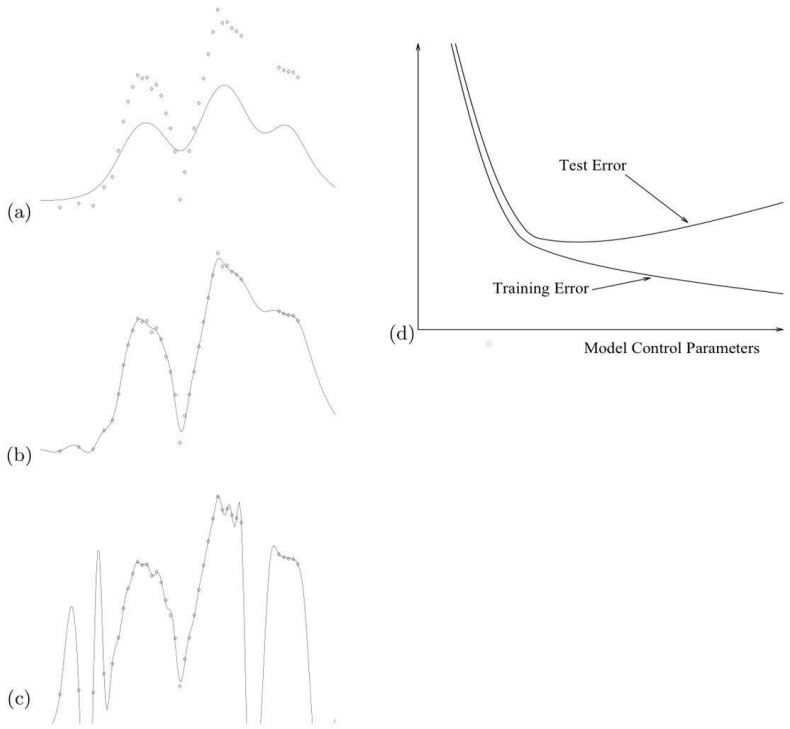



Figura 2.6: Ottimizzazione del modello. Le figure (a-c) illustrano il modello di una funzione che interpola un set di dati composto da una variabile di input e una di output al variare di un certo parametro di controllo. Tale parametro viene via via incrementato per aumentare la complessità del modello (da (a) a (c)) e permettere una maggiore interpolazione dei dati, ma dopo un certo valore si nota che la capacità di predizione ( rappresentato qui dal test error) del modello peggiora. La zona in cui sia il train che il test error (sulla sinistra di (d)) è detta zona di *underfit*, mentre quella a destra di *overfit*. [6]

Per ovviare a questo tipo di problemi e riuscire quindi a stimare l'abilità di una rete a generalizzare, solitamente si calcola la funzione di errore su un set di esempi esterno a quello di train detto *test set*. Confrontando quindi gli errori su questi due insiemi di esempi ci si può quindi fare un'idea delle capacità predittive della propria rete (Fig. 2.6).

Più formalmente, il *test error* non è altro che la media campionaria della funzione rischio vera (sotto l'ipotesi che il test set sia stato generato mediante estrazioni indipendenti dalla distribuzione vera $\mathcal{D}$). Quindi, anche non potendo accedere all'errore vero che sto commettendo col mio predittore (non conoscendo la distribuzione vera dei dati), posso tuttavia darne una stima maggiorandolo col test error. Possiamo inoltre stimare anche lo scarto tra i due errori grazie ad una maggiorazione legata alla legge dei grandi numeri (detta maggiorazione di Chernoff-Hoeffding) [4], ottenendo (fissato un certo $\delta \in (0,1)$)

$$\left|er_{\mathcal{D}}(h) - \widehat{er}(h)\right| \leqslant \sqrt{\frac{1}{2n} ln \frac{2}{\delta}}, \tag{2.17}$$

che vale con probabilità $1-\delta$ rispetto all'estrazione del test set, dove $er_{\mathcal{D}}(h)$ è l'errore vero (o statistico) dato il predittore $h$ calcolato conoscendo $\mathcal{D}$, $\widehat{er}(h)$ è il test error e $n$ è invece la dimensione del test set.

## 2.3 Costruzione dei dataset per l'apprendimento

La costruzione di un buon dataset per l'apprendimento è fondamentale per la riuscita del training della rete, dove per "buono" si intenda un dataset col minor rumore, ossia con la minore presenza di falsi (cioè istanze $\mathbf{x}_n$ a cui è stata assegnata un etichetta $y_n$ sbagliata per via di analisi non corrette o errori accidentali), e con più informazione possibile: notiamo infatti, ancora una volta, che assumiamo che il set di dati che si utilizza per l'apprendimento della rete sia in realtà estratto da una distribuzione $\mathcal{D}$ a noi sconosciuta. Cercare di costruire il training set $\mathcal{S}$ contentente maggiore informazione possibile, significa quindi prenderlo in modo tale che sia il più rappresentativo possibile della distribuzione $\mathcal{D}$.

Una buona parte del mio lavoro di tesi si è quindi concentrata sulla raccolta e l'elaborazione di dati dei due sistemi indagati da usare per l'apprendimento delle reti utilizzate.

### 2.3.1 Proteina G

Spesso le proteine mostrano due proprietà chiave, ossia: hanno una configurazione unica (a bassa entropia) stabile nel loro stato di equilibrio che mantengono per un grande range di temperatura, e mostrano una transizione del primo ordine tra questo stato detto *nativo* e uno stato disordinato, detto *denaturato*. Dato che siamo interessati a studiare un modello in cui

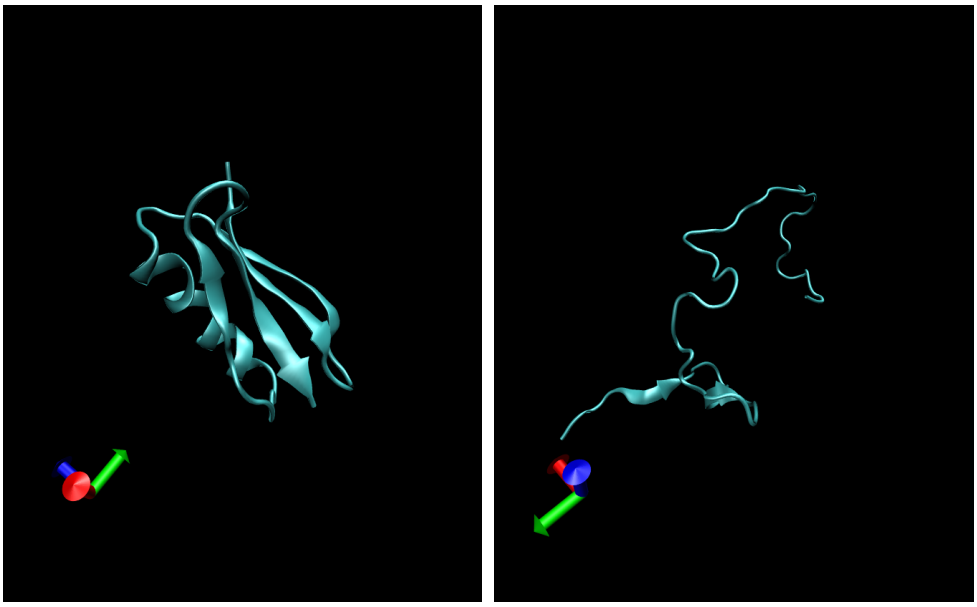


Figura 2.7: Sulla sinistra la configurazione nativa della proteina G mentre sulla detsra un esempio di quelle denaturate.

la proteina studiata, ossia la proteina G, si possa trovare solo in questi due stati, utilizziamo un modello introdotto da Nobuhiro Go che cattura queste proprietà del sistema attraverso un particolare potenziale di interazione tra gli N atomi della proteina [3]. Tale modello presuppone la conoscienza della conformazione nativa della proteina studiata, che denotiamo come $\{r_i^{(nat)}\}$. Possiamo definire la mappa di contatto della conformazione nativa tramite

$$M_{ij} \equiv \Theta\big(d_0 - |r_i^{(nat)} - r_j^{(nat)}|\big) \tag{2.18}$$

dove $\Theta$ è la funzione gradino di Heaveside a $d_0$ è una soglia che definisce i contatti nativi. In un modello Go l'energia potenziale, tipicamente una funzione tipo Lennard-Jones, sarà

$$U(\{r_i\}) = \sum_{i<j} \frac{c_{10}}{|r_i - r_j|^{10}} - \sum_{i<j-3} \frac{M_{ij}}{|r_i - r_j|^6} \tag{2.19}$$

che sarà quindi attrattivo per atomi che sono in contatto nella conformazione nativa e repulsivo altrimenti. Di conseguenza ogni conformazione tipo nativa definirà un minimo del potenziale.

Per la prima parte di questo lavoro si e' quindi simulata la dinamica della proteina G secondo un modello Go alla sua temperatura di transizione tra uno stato e l'altro tramite il programma di dinamica molecolare GROMACS [12] per un intervallo di tempo di 90ns. Tale proteina è formata da

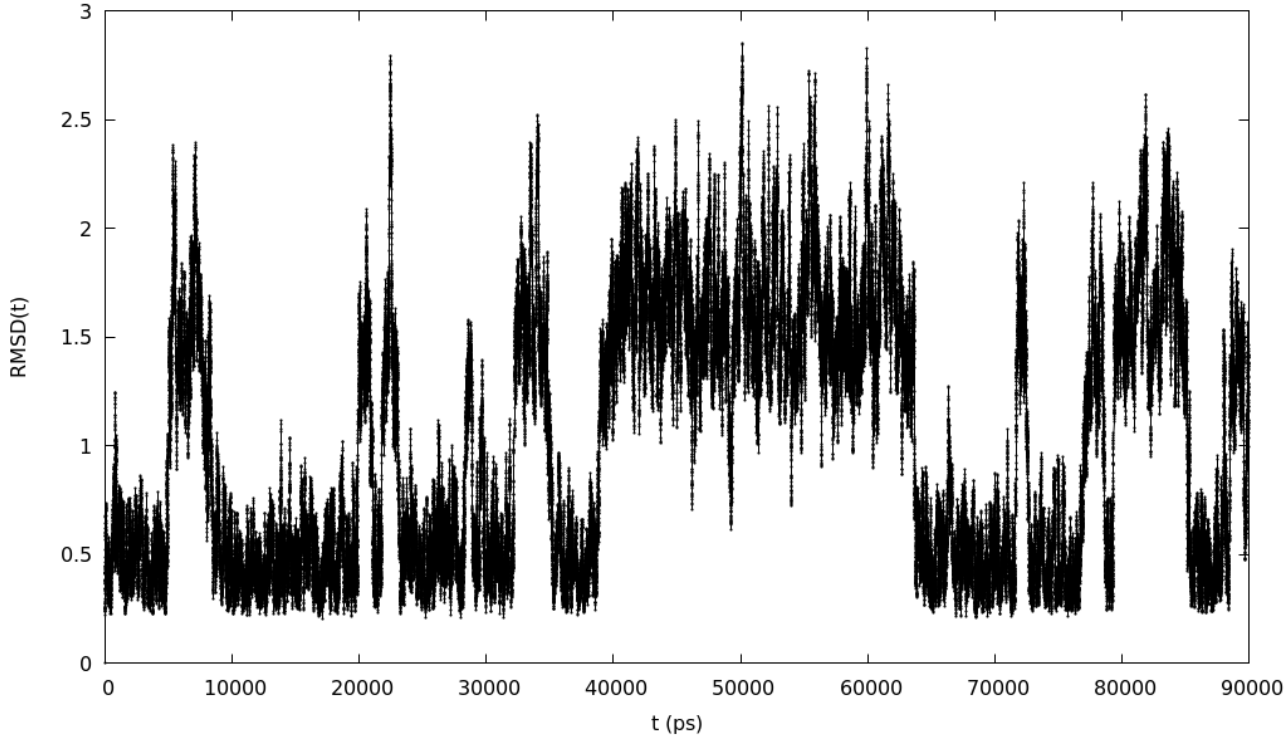



Figura 2.8: RMSD delle configurazioni dei singoli frame rispetto alla configurazione dello stato nativo per la simulazione di 90ns della dinamica della proteina G alla sua temperatura di equilibrio

436 atomi e 56 residui (ossia particolari raggruppamenti di atomi detti anche amino acidi). Solitamente, data la grande abbondanza di dati raccolti sulla sua natura e la sua dimensione abbsatanza ridotta, è utilizzata come modello *toy*, come fatto in questo lavoro.

Calcolando quindi l'RMSD delle singole configurazioni al tempo t rispetto alla configurazione nativa, si nota che la termodinamica del sistema così simulata è effettivamente a due stati (Fig. 2.7). Sempre utilizzando l'RMSD si sono quindi discriminate le conformazioni scegliendo un valore di soglia che separasse i due stati. Basarci sull'RMSD per definire gli stati per poi fare riduzione dimensionale è in effetti un cortocircuito logico, essendo l'RMSD già una variabile ridotta. In realtà questo è solo un test per capire se l'algoritmo riesce almeno a distinguere questi due stati, che data la semplicità del modello Go sono esattamente separati, come si vede in Fig. 4.17.

Avendo estratto una configurazione spaziale per frame della simulazione, ho per prima cosa studiato l' autocorrelazione temporale tra i vari frame basandomi sulla somma mediata dell' RMSD delle coppie di conformazioni distanti $\Delta t$

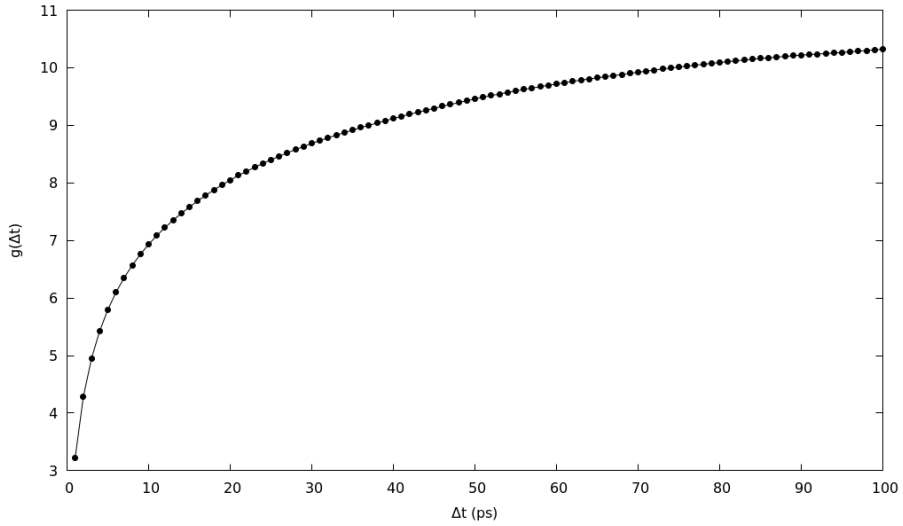



Figura 2.9: Autocorrelazione temporale delle configurazione dei singoli frame estratti dalla traiettoria della dinamica della proteina G con $g(\Delta t)$ calcolata secondo l' equazione (Eq. 2.20).

$$g(\Delta t) = \frac{1}{N} \sum_{t} RMSD(\mathbf{r}(t), \mathbf{r}(t + \Delta t)) \tag{2.20}$$

(Fig. 2.9). In questo modo mi posso fare un'idea di dopo quale intervallo di tempo $\Delta t$ le mie configurazioni risultino scorrelate per capire quanto il mio dataset sia rappresentativo dello spazio delle configurazioni. Infatti, se io considerassi conformazioni prese ad intervalli minori di $\Delta t$, essenzialmente starei prendendo più volte la stessa conformazione. Si nota quindi che le singole configurazioni iniziano ad essere realmente scorrelate temporalmente a distanza di 10 (ps). Tuttavia, data la grande disponibilità di dati, si decide di prenderle tutte tenendone in conto la correlazione temporale per futuri impieghi.

Da queste configurazioni sono state quindi estratte le singole mappe di contatto (come nell'eq (2.18) per la configurazione dello stato nativo) e le distribuzioni dei diedri (coppie di angoli particolari per residuo che individuano univocamente la struttura della proteina) per i singoli frame. Si noti che estraendo e andando a studiare le mappe di contatto si sta già facendo una sorta di riduzione dimensionale del sistema perdendo informazione su di esso: non vi è infatti una corrispondenza $(1 : 1)$ tra mappa di contatto e configurazione. Al contrario, ad ogni distribuzione di diedri corrisponde una e una sola configurazione spaziale.

Per quanto riguarda le mappe di contatto si è scelta una soglia $d_0 = 3.5\mathring{A}$, soglia che permetteva di avere un numero significativo di contatti.

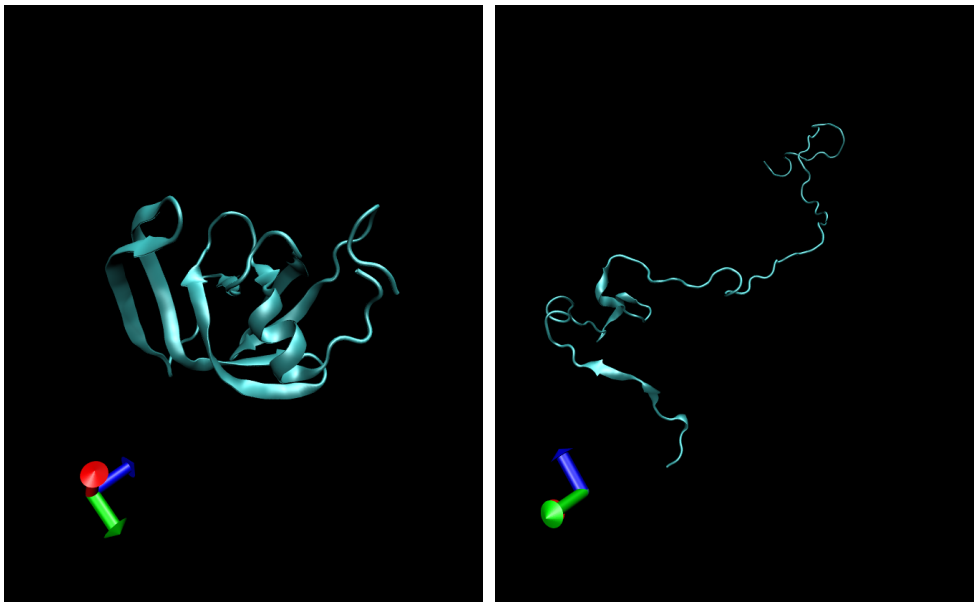


Figura 2.10: Sulla sinistra la configurazione nativa dell'HIV proteasi mentre sulla destra un esempio di quelle denaturate.

Per l'estrazione delle distribuzioni dei diedri si è usato invece un tool di GROMACS chiamato PLUMED [1], che estrae i diedri dai file .pdb (file in cui si dispensano tutte le informazioni sulle coordinate spaziali della proteina) dei singoli frame.

### 2.3.2 HIV proteasi

Si sono studiate inseguito delle conformazioni estratte da traiettorie dell'HIV proteasi, enzima responsabile della riproduzione del virus. Tali traiettorie sta volta non sono state create da me, ma sono frutto di studi di dinamica molecolare precedenti al mio in cui non si usano modelli Go, ma modelli più realistici e complicati che sono in grado di replicare diverse evidenze sperimentali della proteina nel suo stato di equilibrio [5].

Anche in questo caso assumiamo tuttavia che la proteina si possa trovare solo in due stati, foldato e denaturato (Fig. 2.10). Questa volta per distinguere le conformazioni nei due stati ho usato due traiettorie a temperature diffenti: una alla quale il sistema si trovasse sicuramente in uno stato foldato, più bassa, e una più alta alla quale la proteina si trovasse in uno stato denaturato per la maggior parte del tempo. Infatti, data l'ergodicità del sistema, non è detto che non si possa trovare per qualche istante in uno stato nativo. Utilizziamo tuttavia tutte le conformazioni come se fossero appartenenti allo stato denaturato, confidnado nel fatto che le configurazioni veramente appartenenti a questo stato siano molte di più di quelle "false".

Per estrarre le mappe di contatto e le configurazioni dei diedri si sono usate le stesse modalità e gli stessi parametri usati per la proteina G.

# Capitolo 3

# Predizioni sullo stato della proteina tramite percettrone

Partendo dalla architettura più semplice di rete neurale ho iniziato con lo studiare lo stato termodinamico di una proteina G, per poi passare allo studio dello stato di una HIV proteasi. Ho quindi utilizzato sia i dati sulle mappe di contatto che sui diedri.

Per tutta la durata del lavoro ho scritto gli algoritmi necessari di reti neurali tramite l'impiego di librerie scritte in Python ( chiamate *Pytorch* - PYTORCH [2]).

## 3.1 La rete utilizzata: il percettrone

Per studiare lo stato termodinamico dei due sistemi, ho costruito un percettrone, ossia una rete neurale con un solo nodo nello strato di output e nessuno strato nascosto (Fig. 3.1), con

$$\sigma(a_{1,1}) = \frac{1}{1 + exp(-a_{1,1})} \tag{3.1}$$

dove,

$$a_{1,1}(\mathbf{x}) = \sum_{i=0}^{n} w(v_{0,i}, v_{1,1}) x_i. \tag{3.2}$$

Si è utilizzato inoltre l'algoritmo (SGD) come algoritmo di apprendimento e l'errore quadratico medio come funzione costo. Seppur infatti l'output aspettato, sia per le mappe di contatto che per i diedri, è "*0*" per le configurazioni native e "*1*" per quelle denaturate, il valore dell' output predetto è un numero reale $\in [0, 1]$ (dato che ho scelto $\sigma$ (Eq. 3.1) come funzione di attivazione per il nodo in output).

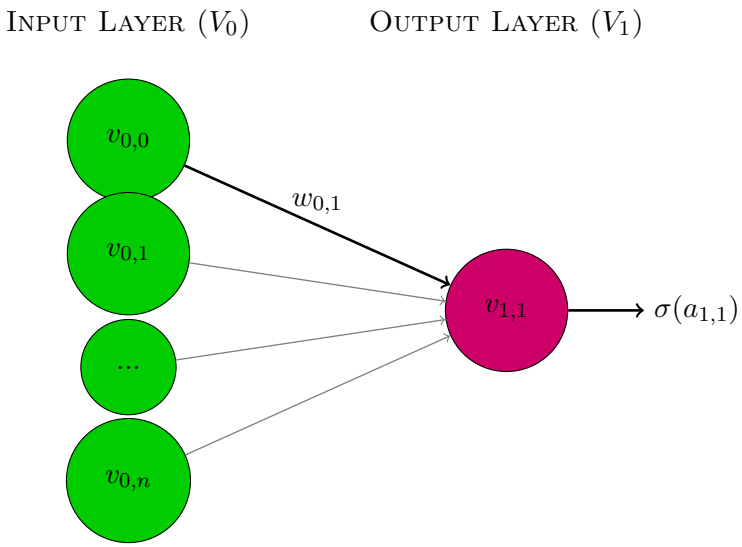



Figura 3.1: Esempio di percettrone con $(n+1)$ nodi in input, $w_{i,1} \in W$ che ha dimensione $((n+1) \times 1)$ e $\sigma$ funzione di attivazione del singolo nodo in output.

Quello che fa questa rete è quindi un fit pesato non-lineare della funzione $h(w_{ij}; \mathbf{x}_t)$ sul dataset $(n+1)$ dimensionale (dove con n si intende la dimensione dell'input) $S_t = \{(\mathbf{x}_{(1)}, y_{(1)}), ..., (\mathbf{x}_{(t)}, y_{(t)})\}$, minimizzando la funzione costo empirica $l_{\mathcal{S}_t}(h(w_{ij}; \mathbf{x}_t), y_t)$ .

## 3.2 Proteina G

### 3.2.1 Mappe di contatto

Per quanto riguarda le mappe di contatto della proteina G si hanno quindi $n = 1540$ input, pari agli elementi della triangolare superiore della matrice simmetrica che individua la mappa dei contatti. Utilizzo un training set composto da $t = 105000$ esempi, in modo da avere un numero di punti molto maggiore rispetto al numero di variabili (W ha infatti dimensioni $(1541 \times 1)$).

Ho quindi lanciato l'algoritmo di apprendimento con diverse epoche per studiarne lo stato. Si sono inoltre prodotte 10 repliche delle singole run (data la randomizzazione iniziale dei pesi non si avranno infatti sempre risultati identici). Nel grafico (Fig. 3.2) si può notare come già con due sole epoche la rete riesca, in media, a distinguere i due stati con un errore quadratico medio minore di 0.25; ciò significa che mediamente lo scarto tra lo stato predetto e quello vero è sempre minore di 0.5. Quindi quello che leggiamo da questo grafico è che, se arrotondiamo a 0 e 1 l'output una volta predetto

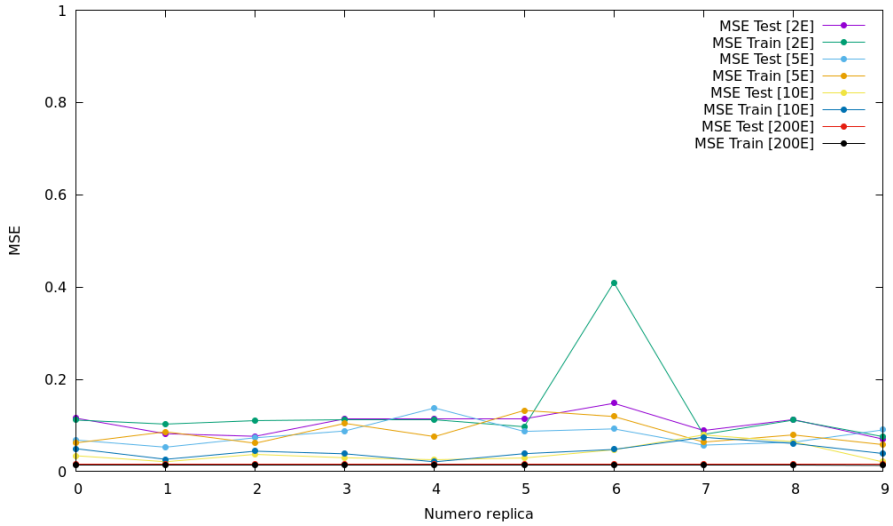



Figura 3.2: MSE delle repliche delle reti addestrate con mappe di contatto della proteina G per diverse epoche (E).

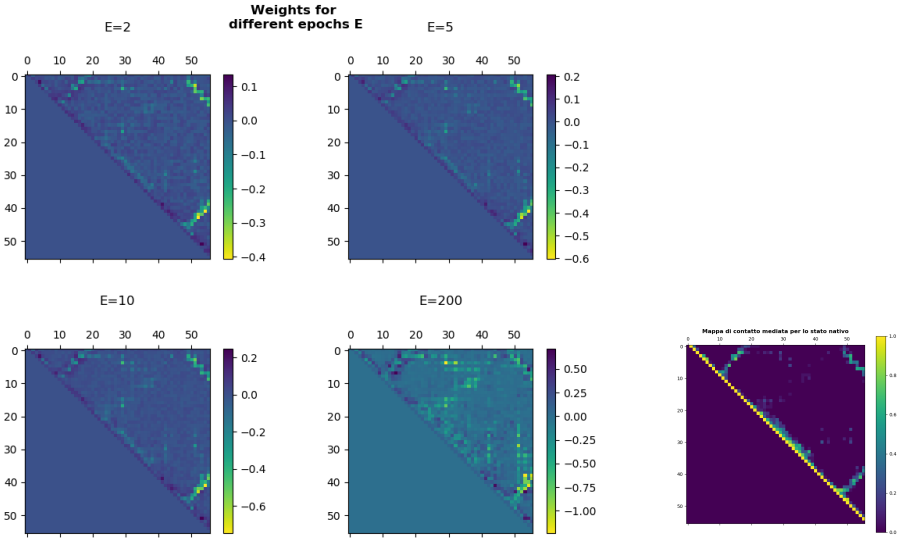



Figura 3.3: Sulla sinistra le heatmap dei pesi di una replica estratta dagli ensamble a diverse epoche (E). Sulla destra la heatmap della media di 20 configurazioni dello stato nativo della proteina G. Si noti che nelle mappe di sinistra i colori nella barra di colore sono stati invertiti rispetto a quella di destra per mettere in risalto le strutture secondarie.

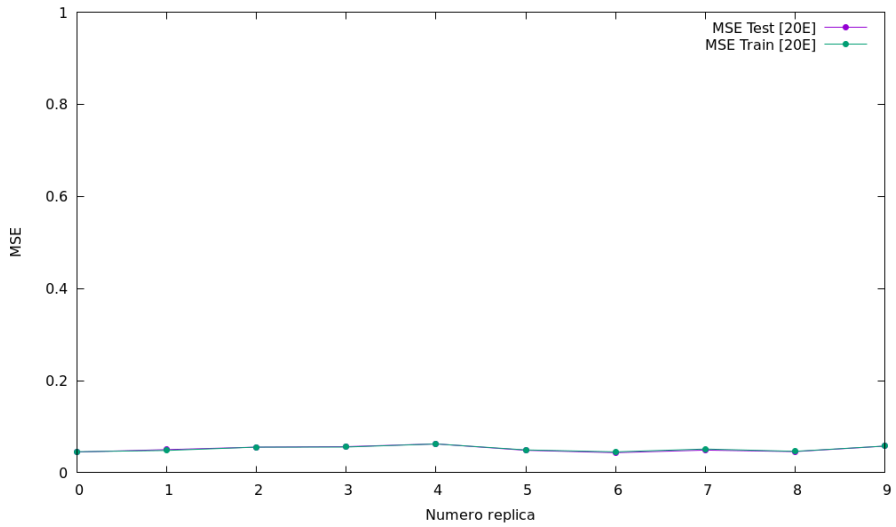



Figura 3.4: MSE delle repliche della rete addestrata con le distribuzioni dei diedri della proteina G per 20 epoche.

dalla rete, mediamente sbaglieremo per un numero di volte molto prossimo a 0, se non identicamente nullo.

Per capire se la rete stesse davvero imparando a distinguere i due stati, ho studiato i pesi di una delle repliche per caso con diverse epoche, cercando di capire quali contatti influissero di più nella predizione. Ho quindi disegnato la matrice dei pesi sotto forma di mappa di calore (Fig. 3.3, sx). Confrontando questi grafici con la *heatmap* della media di 20 configurazioni dello stato nativo (Fig. 3.3, dx), si nota che i pesi più alti, quindi più influenti, si trovano in corrispondenza dei contatti che individuano le strutture secondarie della proteina. Questa evidenza rispetta ciò che ci si potrebbe aspettare: quello che differenzia la configurazione di uno stato nativo da quella di uno stato denaturato è la formazione di queste strutture e quindi la presenza di contatti tra di esse.

### 3.2.2 Diedri

Per le distribuzioni dei diedri si hanno $n = 110$ input. Anche in questo caso faccio in modo di avere un numero di esempi $t$ di molto superiore al numero di variabili (questa volta W ha dimensioni $(111 \times 1)$).

Fissato un numero di epoche ragionevole (E = 20), ho ottimizzato 10 repliche. Come si può vedere nel grafico (Fig. 3.4), le 10 repliche hanno un errore quadratico medio molto sotto la soglia dello 0.25; mi aspetto quindi di avere, come per le mappe di contatto, un numero di predizioni sbagliate praticmente nullo tutte le volte.

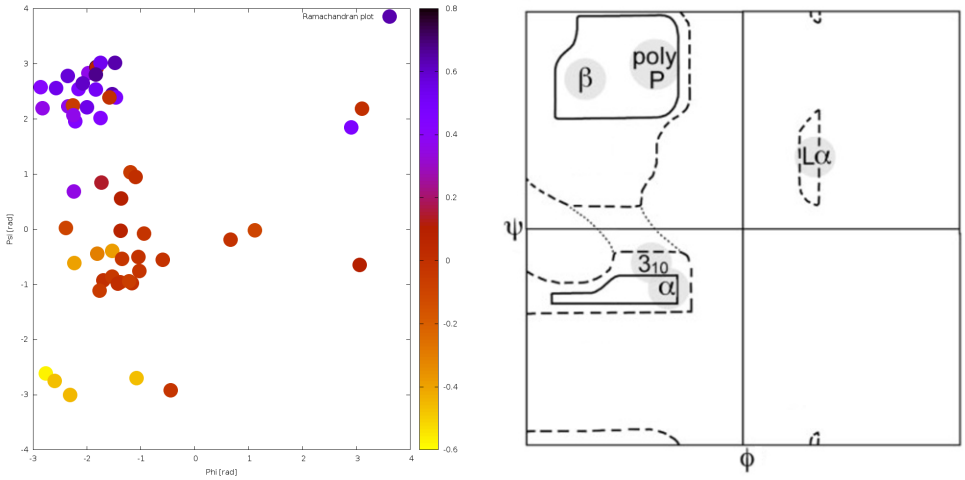



Figura 3.5: Sulla sinistra si hanno i diedri dei 55 residui della proteina G il cui colore indica il valore del peso associato. Sulla destra si ha il grafico di Ramachandran che visualizza le zone $(\phi, \psi)$ corrispondenti alle diverse strutture secondarie. Si noti che i pesi della rete corrispondenti ai diedri della proteina che individuano le strutture $\beta$ ( formate nello stato nativo) sono quelli che influiscono di più.

Anche per i diedri cerco un altro modo per capire se la rete sta imparando a distinguere i due stati. Costruisco quindi il grafico di Ramachandran per la distribuzione di diedri dello stato nativo della proteina G a cui associo un mezzo volte la somma dei due pesi corrispondenti ai due diedri $\frac{1}{2}(\phi_i + \psi_i)$ (Fig. 3.5, sx). Andandolo a confrontarre con il grafico aspettato (Fig. 3.5, dx), si noti come i pesi che influiscono di più siano quelli in corrispondenza dei diedri che individuano le strutture secondarie.

## 3.3 HIV proteasi

### 3.3.1 Mappe di contatto

Nel caso della HIV proteasi si hanno più residui (99 per l'esattezza), più contatti ($n = 4851$) e quindi più variabili (W ha dimensioni $(4852 \times 1)$); si tratta quindi di un sistema più complicato. Anche in questo caso tuttavia si riescono a raccogliere un numero di esempi maggiore del numero di variabili.

Studio quindi l'abilità predittiva dello stato termodinamico della proteina della rete per diverse epoche, come fatto per la proteina G. Nel grafico (Fig.3.6) vediamo che mediamente le repliche riescono a distinguere i due stati come discusso per la proteina G, seppur con più difficoltà rispetto a prima.

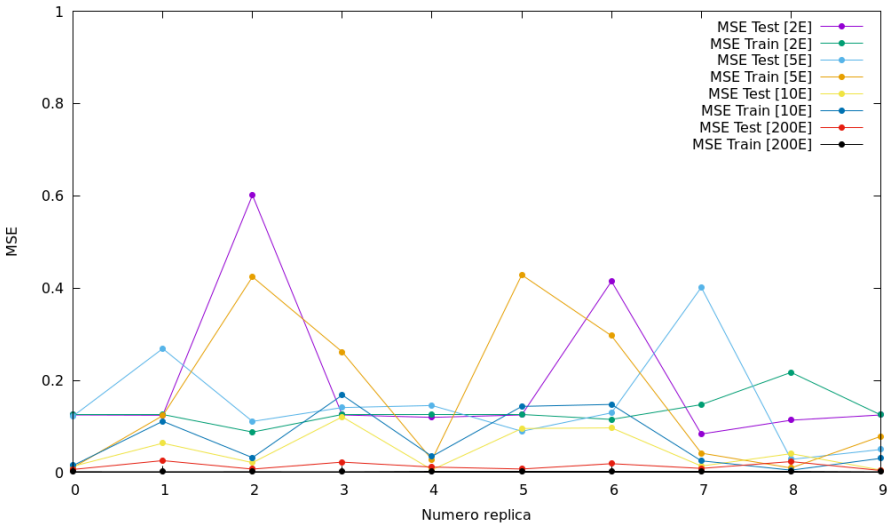



Figura 3.6: MSE delle repliche delle reti addestrate con mappe di contatto della HIV proteasi per diverse epoche.

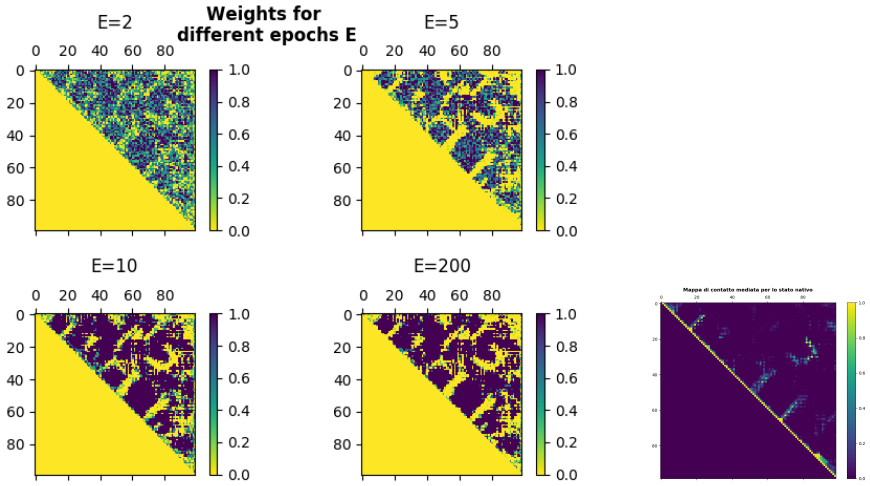



Figura 3.7: Sulla sinistra le Heatmap dei pesi di una replica estratta dagli ensamble a diverse epoche. Sulla destra la heatmap della media di 20 configurazioni dello stato nativo dell'HIV. Anche qui vale lo stesso discorso fatto sui colori delle heatmap per la proteina G.

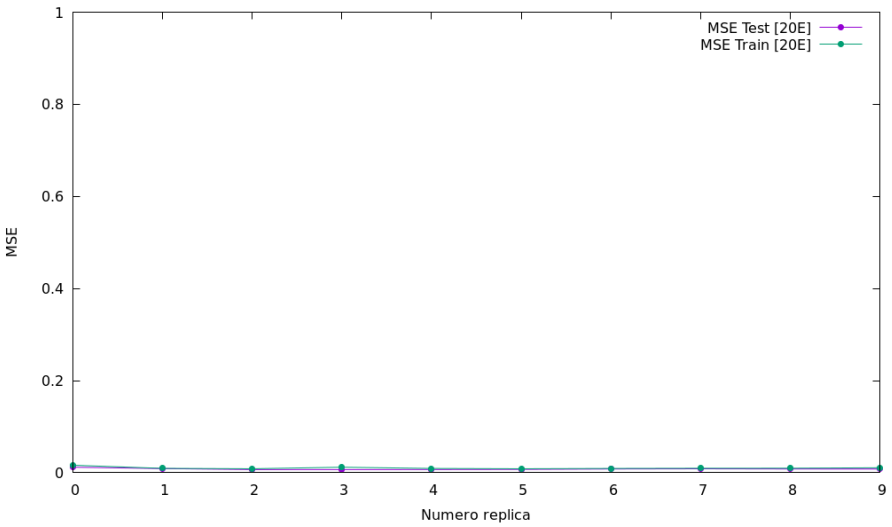



Figura 3.8: MSE delle repliche della rete addestrata con distribuzioni della HIV proteasi per 20 epoche.

Andando a studiare i pesi della rete una volta addestrata come fatto per la proteina G, vediamo che anche qui i pesi più rilevanti si trovano in corrispondenza dei contatti che individuano le strutture secondarie formate nello stato nativo (Fig.3.7 )

### 3.3.2 Diedri

Infine impiego lo stesso tipo di rete utilizzando finora anche con le distribuzioni dei diedri dell'HIV proteasi. Si hanno quindi $n = 196$ input mentre W che ha dimensioni $(197 \times 1)$. Ancora una volta si utilizza un numero molto maggiore di esempi rispetto al numero di variabili.

Dalla Fig. 3.8 si vede che tutte le repliche mediamente distinguono i due stati come discusso nei casi precedenti.

Costruisco anche per L'HIV proteasi il grafico di Ramachandran per la conformazione nativa a cui associo, allo stesso modo di prima, i pesi corrispondenti dei due diedri (Fig. 3.9). Si nota ancora una volta che i pesi corrispondenri ai diedri delle strutture secondarie sono quelle che influiscono di più nella predizione.

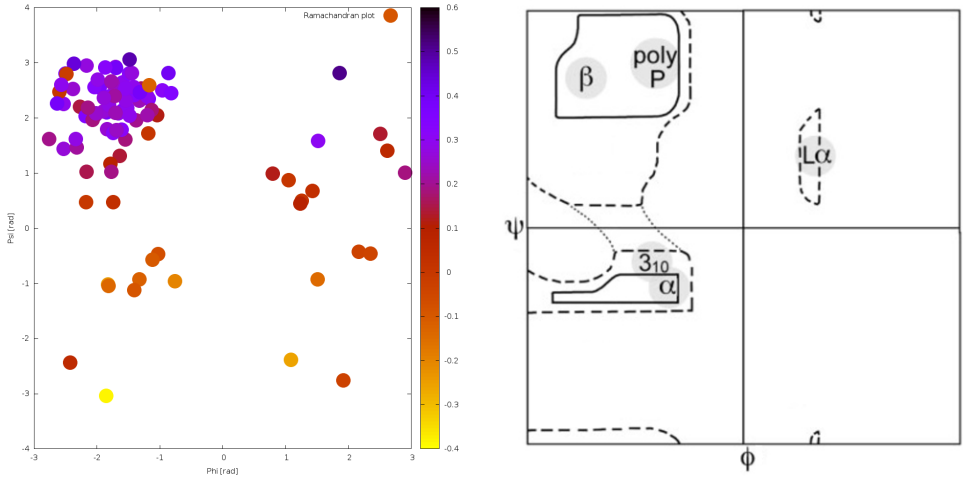



Figura 3.9: Sulla sinistra si hanno i diedri dei 98 residui dell' HIV proteasi il cui colore indica il valore del peso associato. Sulla destra si ha ancora il grafico di Ramachandra.

# Capitolo 4

# Riduzione dimensionale tramite Autoencoder

Gli autoencoder sono particolari architetture di reti neurali artificiali dove l'input coincide con l'output. Ho cercato quindi di trovare una particolare architettura della rete che mi permettesse di codificare in modo corretto la termodinamica del sistema studiato, operando quindi una riduzione dimensionale. Sono quindi partito dallo studiare come veniva codificata l'infomazione di distribuzioni conosciute di mappe di contatto e distribuzioni di diedri sintetiche. In seguito, sono passato a studiare le configurazioni vere di proteina G e HIV proteasi estratte da distribuzioni di Boltzman in spazi complicati altodimensionali.

Ho infine concluso il mio lavoro con un primo studio di una architettura di autoencoder con più strati nascosti.

## 4.1 Gli autoencoder

Per questa parte di lavoro ho utilizzato una particolare architettura di rete neurale conosciuta come *autoencoder*. Tale rete ha la particolarità di avere lo stesso numero di nodi in input e in output; durante l'apprendimento vengono quindi somministrati gli input uguali all'output (si ha un training set della forma $\mathcal{S}_n = \{\mathbf{x}_n, \mathbf{x}_n\}$). In questo modo si costringe la rete a riprodurre esattamente ciò che le viene dato in pasto, passando per uno strato intermedio che solitamente ha un numero minore di nodi (Fig. 4.1).

Dallo strato di input si passa quindi allo strato nascosto codificando l'infomazione in entrata in uno spazio con delle dimensioni ridotte. Passando invece dallo strato nascosto a quello di output si decodificano le variabili dell'hidden ritornando nello spazio di partenza.

A seconda della complessità del sistema indagato si sono quindi usate funzioni di attivazione lineari e sigmoidi per i nodi dello strato nascosto. Come algoritmo di apprendimento viene ancora una volta impiegato (SGD).

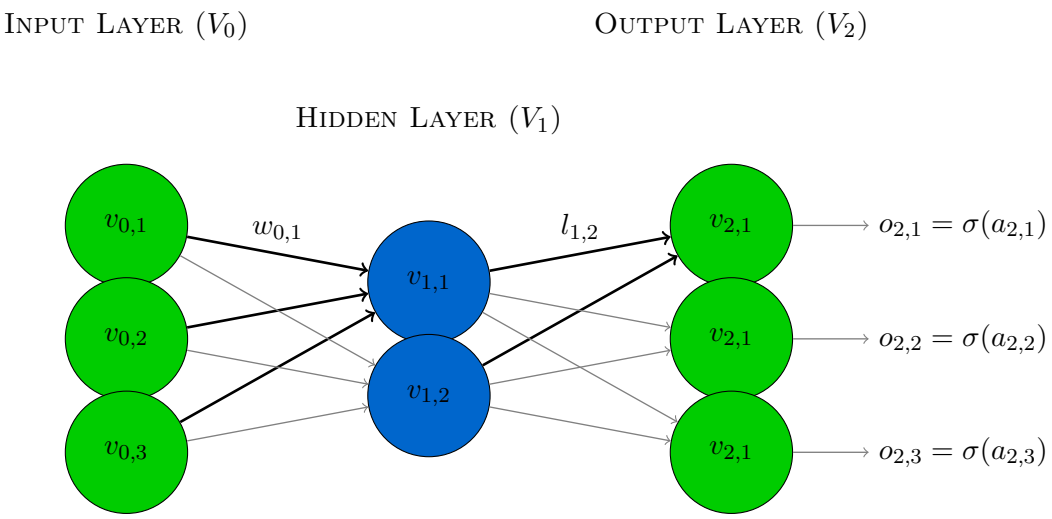



Figura 4.1: Esempio di autoencoder con un solo hidden layer. Nella parte sinistra della rete avviene la codifica dell'input nei nodi dell'hidden. Nella parte destra invece si ha la decodifica, dove quindi il valore dei nodi in hidden viene rimappato in output per riprodurre i valori dell'input.

Si noti che questo tipo di algoritmo non fa più parte degli algoritmi di apprendimento automatico supervisionato (ossia algoritmi per cui avevamo training set del tipo $\mathcal{S}_n = \{\mathbf{x}_n, y_n\}$). Non stiamo infatti dando alcuna etichetta alle configurazioni che passiamo alla rete in input. Vogliamo vedere se riesca a codificare in modo differente conformazioni di stati differenti nell'attivazione dei nodi in hidden. Questi algoritmi vengono detti di apprendimento *non*-supervisionato.

## 4.2 Studio con le mappe sintetiche

Sono quindi partito dallo studiare mappe di contatto e distribuzioni di diedri sintetiche (ossia mappe e diedri che ho costruito personalmente e che non sono quindi state estratte da nessuna distribuzione caratterizzata da una particolare energia) per ottenere qualche intuizione in più sulle capacità di codifica della rete in base alla sua architettura. Ciò che mi aspetto è che: reti con un architettura lineare (ossia con una funzione di attivazione lineare per i nodi dell'hidden layer) riescano a distinguere solo conformazioni differenti per via di qualche rototraslazione (nello spazio delle mappe di contatto e dei diedri, non nello spazio reale), mappabile quindi linearmente da una matrice. Mentre reti non lineari siano in grado di distinguere conformazioni che si differenziano per trasformazioni non lineari. Inoltre mi aspetto di avere una certa correlazione tra numero di nodi in hidden e numero di stati: se prendo infatti un' architettura con funzione di attivazione sigmoide, posso

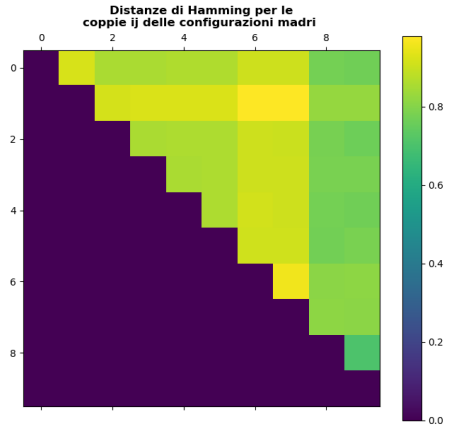



Figura 4.2: Heatmap delle distanze di Hamming tra le 10 mappe di contatto sintetiche. Si noti che le 8 e 9 sono le meno distanti poichè sono quelle con più contatti di tutte.

pensare ai due nodi come a dei bit in cui viene codificata l'informazione in input. Infatti, più la funzione sigmoide sarà ripida più somiglierà ad una funzione soglia (funzione con cui possiamo rappresentare un bit). Mi aspetto quindi di poter codificare correttamente almeno $2^n$ stati differenti avendo a disposizione $n$ bit.

Ho quindi costruito dieci diverse mappe di contatto sintetiche sul modello di quelle della proteina G: per una ho proprio preso la mappa di contatto della conformazione nativa della proteina G; per le altre nove ho generato invece un vettore di $m$ numeri casuali uniformemente distribuiti nell'intervallo $[1, 1540]$, dove $m$ è il numero di contatti da inserire all'interno della mappa di contatto, mentre il valore di ogni elemento di questo vettore m-dimensionale corrisponde all'indice dell'elemento della mappa di contatto sintetica in cui inserire il contatto. Ho quindi generato le diverse mappe sintetiche variando $m$ (la Fig. 4.2 da un'idea di quanto siano diverse le une dalle altre attraverso una misura delle distanze di Hamming). Ho infine generato dei dataset da 2000 mappe di contatto, uno per ogni mappa sintetica, tramite un algoritmo che ad ognuno degli elementi della matrice applicasse una probabilità del 5% di flip (da 0 a 1 o viceversa).

Ho costruito inoltre dieci distribuzioni di diedri differenti, sempre sul modello della proteina G: per due ho proprio preso la distribuzione dei diedri dello stato nativo e una dello stato denaturato della proteina G; per le altre otto invece ho generato un vettore di 110 elementi (tanti quanto i diedri) dove ogni elemento corrisponde a un numero estratto da una distribuzione uniforme nell'intervallo $[-\pi, \pi]$ (la Fig. 4.3 da un'idea di quanto siano diverse le une dalle altre attraverso una misura dell'MSE). Ho infine generato dei

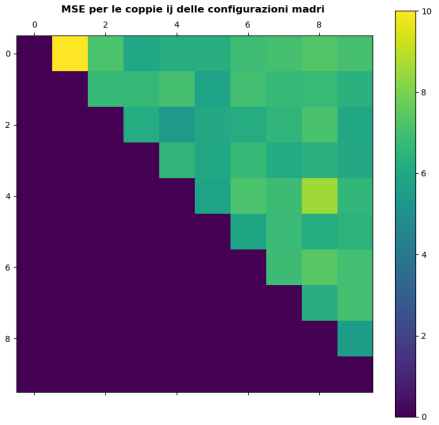
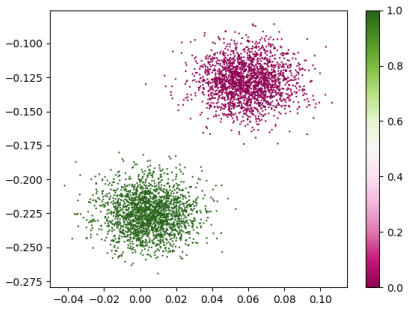



Figura 4.3: Errore quadratico medio (MSE) clacolato per le diverse coppie di distribuzioni di diedri. Si noti che si ha un MSE poco oscillante poichè quasi tutte le configurazioni sono estratte in modo casuale(a parte per la 0esima con la prima che sono le più lontane, dato che corrispondono alle distribuzioni di uno stato nativo e denaturato).

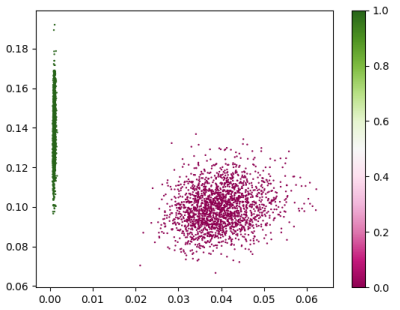
dataset da 2000 distribuzioni di diedri, uno per ogni configurazione sintetica, sommando ad ogni elemento dei vettori 110-dimensionali un numero estratto da una distribuzione gaussiana centrata sull'elemento stesso entro un $\sigma$.

Studio quindi come viene codificata l'informazione in ingresso andando a visualizzare l'attivazione dei nodi nello strato nascosto. Si noti che si sono utilizzati massimo 3 nodi in hidden poichè poi non sarebbe stato possibile visualizzare l'attivazione dei nodi per verificare il corretto funzionamento della rete. Ho quindi proceduto per gradi andando a studiare, sia per le mappe di contatto che per i diedri, dataset via via sempre più complicati (ossia con più configurazioni sintetiche) aumentando passo passo la complessità della rete.

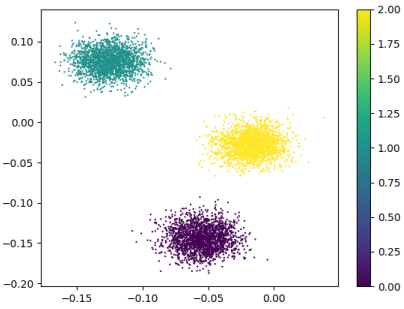
Nelle Fig. 4.4 e Fig. 4.5, dove ogni rete ha esattamente due nodi nello strato nascosto (di cui vediamo disegnata l'attivazione), si nota come, contrariamente a ciò che ci aspettavamo, le reti con una funzione di attivazione sigmoide non mostrano una capacità di classificazione nettamente maggiore rispetto a quelle lineari. Dalla Fig. 4.4f, si nota tuttavia che non vengono classificate tutte le configurazioni sintetiche ma comunque si ha una divisione che rispetta le distanze di hamming: come si vede dalla Fig. 4.2, le due mappe più simili a tutte le altre (quelle con più contatti di tutti, ossia la 8 e la 9, nelle due tonalità più scure di verde nella Fig. 4.4f) vengono mappate in classi più lontane, mentre le altre, equidistanti tra di loro, vengono messe in una terza classe. Questo potrebbe significare che la rete non è abbastanza



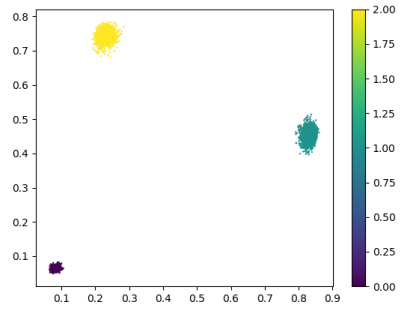
(a) Rete lineare con due mappe di contatto sintetiche.



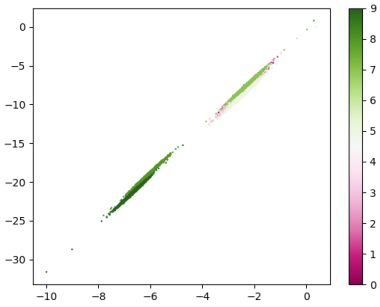
(b) Rete sigmoide con due mappe di contatto sintetiche.



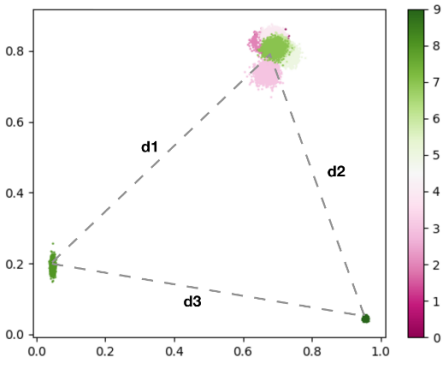
(c) Rete lineare con tre mappe di contatto sintetiche.



(d) Rete sigmoide con tre mappe di contatto sintetiche.

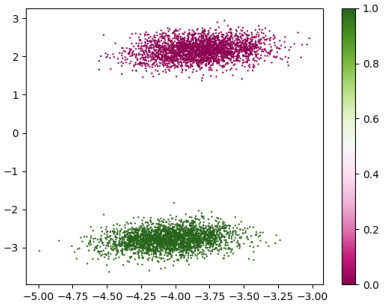


(e) Rete lineare con dieci mappe di contatto sintetiche.

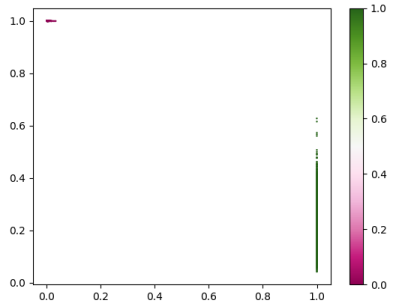



(f) Rete sigmoide con dieci mappe di contatto sintetiche.

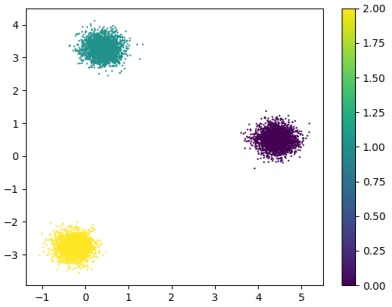
Figura 4.4: Grafici delle attivazioni dei nodi in hidden delle reti con esattamente due neuroni nello stato nascosto. I colori, aggiunti in un secondo momento, indicano le classi di appartenenza.



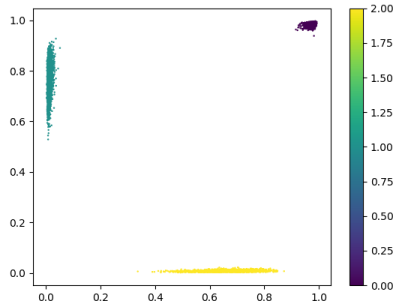
(a) Rete lineare con due distribuzioni di diedri sintetiche.



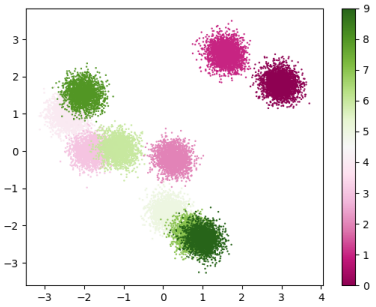
(b) Rete sigmoide con due distribuzioni di diedri sintetiche.



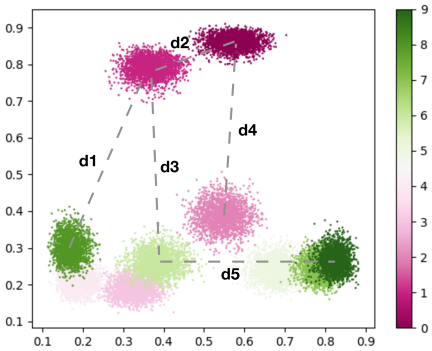
(c) Rete lineare con tre distribuzioni di diedri sintetiche.



(d) Rete sigmoide con tre distribuzioni di diedri sintetiche.



(e) Rete lineare con dieci distribuzioni di diedri sintetiche.




(f) Rete sigmoide con dieci distribuzioni di diedri sintetiche.

Figura 4.5: Grafici delle attivazioni dei nodi in hidden delle reti con esattamente due neuroni nello stato nascosto. I colori, aggiunti in un secondo momento, indicano le classi di appartenenza.

complessa per distinguerle tutte e dieci, ma si limita a classificare solo quelle più diverse.

Dalle figure Fig. 4.5e-4.5f si nota una cosa simile per i diedri: come si vede dala Fig. 4.3, si ha che le due classi corrispondenti alle configurazioni più distanti tra loro secondo l'MSE (la 0 e la 1, corrispondenti a stato nativo e denaturato della proteina G e alle due tonalità di rosso più scure nelle due Fig. 4.5e-4.5f) vengono separate dalle altre, le quali sono equidistanziate tra loro. Tuttavia, come si può notare dagli esempi di distanze disegnate nelle Fig. 4.5e-4.5f, la proiezione nello spazio delle attivazioni non rispetta le distanze relative tra le conformazioni: ad esempio, si noti che secodo l'MSE la configurazione nativa e denaturata sono le più distanti, cosa non più vera nello spazio delle attivazioni (si guardi "d2" nella Fig. 4.5f).

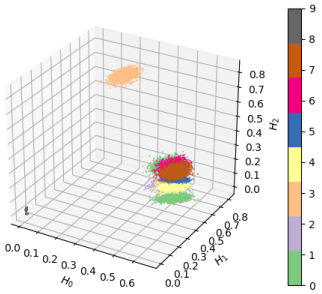
Per vedere tutte e dieci le classi, si potrebbe quindi pensare di aumentare la complessità della rete andando a incrementare magari il numero di strati nascosti (dato che come si vede ad esempio dalle Fig. 4.6a-4.6b non basta incrementare il numero di nodi). Si noti infine, osservando le (Fig. 4.4e-4.4f) che viene rispettata l'altra nostra intuizione: infatti tutte le reti (aventi due nodi nello strato nascosto) riescono a distinguere tutte le classi fino a che con 10 configurazioni non si supera la soglia di 4 ($2^2$) aspettata.

Si prova quindi a stressare al massimo la rete, cercando di fargli distinguere otto conformazioni sintetiche diverse con 3 nodi, in modo tale da avere un egual numero di classi e di configurazioni dei bit ($2^3$) (Fig. 4.6). Anche qui l'architettura sigmoide non dimostra particolari capacità superiori a quelli della rete lineare. Si nota quindi l'impossibilità di distinguere tutte e otto le classi. Ottimizzando 10 repliche della rete sigmoide con i diedri e studiandone l'MSE (Fig. 4.7), si arriva alla conclusione che la rete non abbia problemi di ottimizzazione (dato che tutte le repliche convergono tutte più o meno allo stesso valore e mostrano tutte l'impossibilità di distinguere tutte le classi), ma proprio di architettura. Crolla così anche l'ultima delle nostre intuizioni. Si ha quindi a che fare con dinamiche più complicate e oscure per quanto riguarda la codifica dell'informazione.

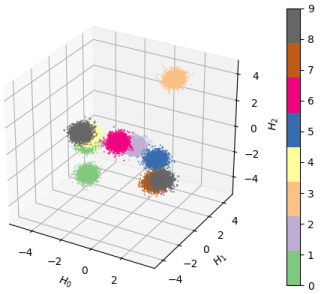
Si passa quindi allo studio delle conformazioni vere aspettandoci di riuscire a risolvere in classi le conformazioni più differenti corrispondenti ai diversi stati termodinamici.

## 4.3 Proteina G

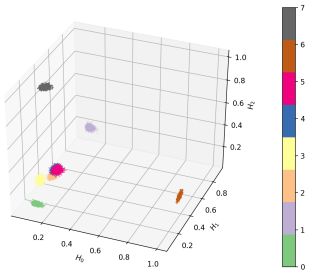
Sono quindi passato allo studio delle diverse conformazioni estratte dalla traiettoria della proteina G. Questa volta quindi non sappiamo come sia fatto lo spazio delle fasi vero: estraiamo infatti conformazioni che seguono una distribuzione di Boltzman in uno spazio complicato alto dimensionale. Applicando il nostro metodo, vogliamo quindi vedere se accade ciò che ci aspettiamo: sapendo che esistono almeno due stati (nativo e denaturato), ci



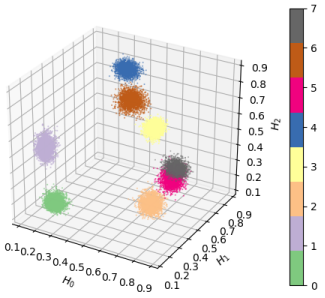
(a) Rete sigmoide con dieci mappe di contatto sintetiche.



(b) Rete sigmoide con dieci distribuzioni di diedri sintetiche.



(c) Rete sigmoide con otto mappe di contatto sintetiche.



(d) Rete sigmoide con otto distribuzioni di diedri sintetiche.

Figura 4.6: Grafici delle attivazioni dei nodi in hidden delle reti con esattamente tre neuroni nello stato nascosto. Come si può vedere, in nessuno dei casi vengono distinte tutte le conformazioni.

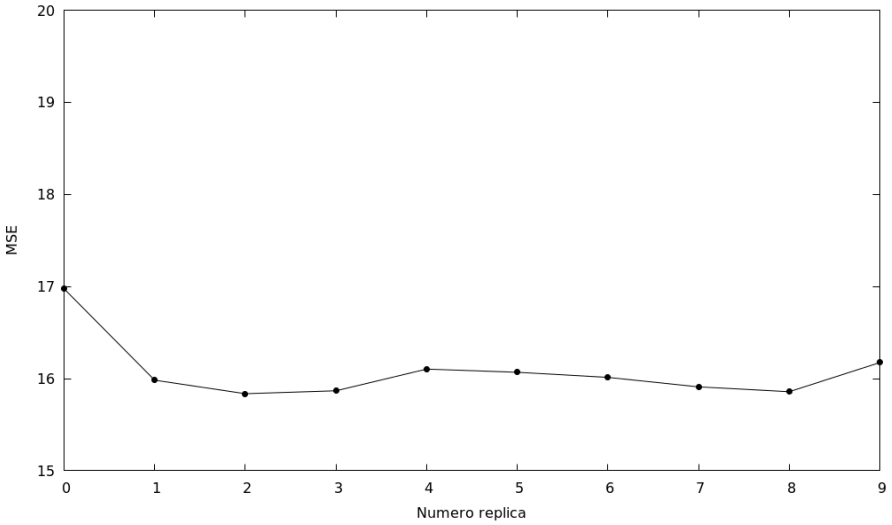


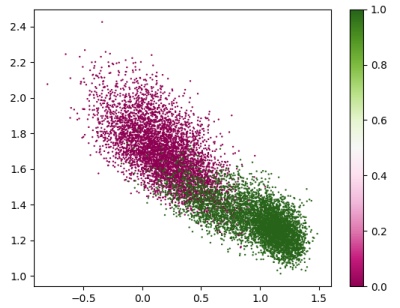

Figura 4.7: MSE delle repliche della rete addestrata con otto distribuzioni sintetiche e tre nodi in hidden. L'errore risulta così alto poichè ora è un errore medio cumulativo, ossia l'errore totale dei 110 output è la somma dell'errore dei singoli output. Andando a dividere per il numero di output avremmo invece l'errore quadratico medio, medio.

aspettiamo che vengano distinte due classi. Non è detto però che esistano solamente due stati; potremmo quindi individuare più classi corrispondenti a stati intermedi.

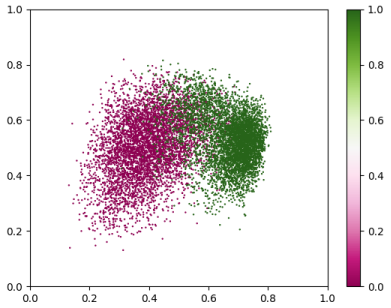
Dalla (Fig. 4.8) vediamo che la rete, come ci aspettavamo, sta cercando di dividere le conformazioni in base al loro stato di appartenenza (i colori, che identificano lo stato delle singole conformazioni, verde per quelle denaturate e rosso per quelle native, vengono infatti inseriti una volta che la rete è già stata addestrata per capire se le due classi corrispondessero ai due stati). Si può affermare quindi che le due nuove variabili, ossia i due nodi nello strato nascosto, stiano riproducendo con successo la termodinamica del modello a due stati della proteina (per lo meno per quanto riguarda le mappe di contatto (Fig. 4.8a-4.8b)).

Si osserva una differenza sostanziale tra le reti che codificano le mappe di contatto e le distribuzioni di diedri: se le prime infatti vengono divise in due classi sufficientemente separate, le seconde rimangono alquanto sovrapposte. Ciò potrebbe essere imputabile al fatto che piccole variazioni dei diedri corrispondono a grandi cambi della conformazione spaziale (cosa invece non vera per le mappe di contatto): per questo motivo la rete non è quindi in grado di distinguere i due stati.

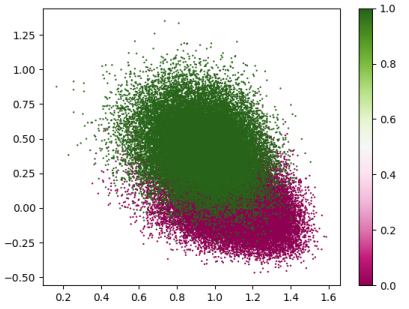
Anche in questo caso non si notano differenze sostanziali tra le reti con architettura lineare e non.



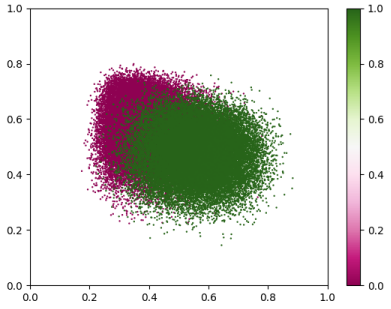

(a) Rete lineare per le mappe di contatto della proteina G.




(b) Rete sigmoide per le mappe di contatto della proteina G.

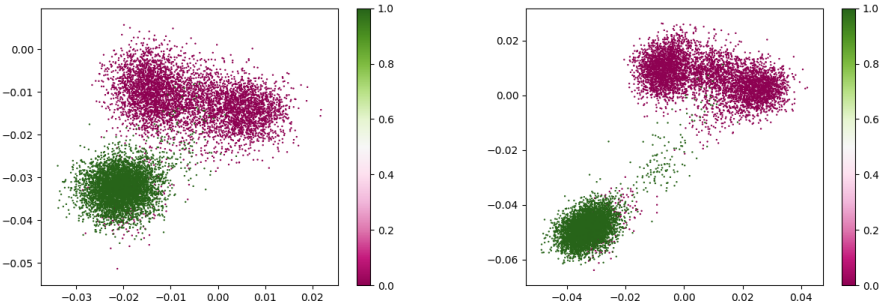



(c) Rete lineare per le distribuzini di diedri della proteina G.



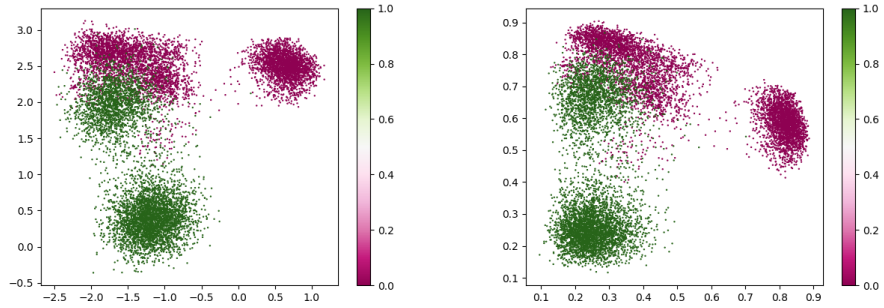

(d) Rete sigmoide per le distribuzini di diedri della proteina G.

Figura 4.8: Grafici delle attivazioni dei nodi in hidden delle reti con esattamente due neuroni nello strato nascosto. I colori, aggiunti in un secondo momento, indicano lo stato di appartenenza (rosso per lo stato nativo, verde per quello denaturato).



(a) Rete lineare per le mappe di contatto dell' HIV.

(b) Rete sigmoide per le mappe di contatto dell' HIV.



(c) Rete lineare per le distribuzini di diedri dell' HIV.

(d) Rete sigmoide per le distribuzini di diedri dell' HIV.

Figura 4.9: Grafici delle attivazioni dei nodi in hidden delle reti con esattamente due neuroni nello stato nascosto. I colori, aggiunti in un secondo momento, indicano la simulazione di appartenenza (una a temperatura più alta (verde) e una più bassa(rossa)).

## 4.4 HIV proteasi

Cerco quindi di studiare anche le conformazioni dell'HIV con lo stesso metodo utilizzato finora.

Per le mappe di contatto (Fig. 4.9a-4.9b) si osserva innanazitutto che entrambe le architetture riescono a distinguere due classi, una per stato. Probabilmente anche l'architettura lineare sta funzionando perchè riesce ad individuare dei contatti nativi, non presenti nelle conformazioni denaturate, e classifica le varie mappe in base alla presenza o meno di alcuni di essi. Questa tattica di riconoscimento infatti non richiederebbe alcun tipo di trasformazioni non lineari.

Per quanto riguarda invece le distribuzioni dei diedri (Fig. 4.9c-4.9d), si osserva la separazione delle distribuzioni in tre classi distinte. Ho quindi

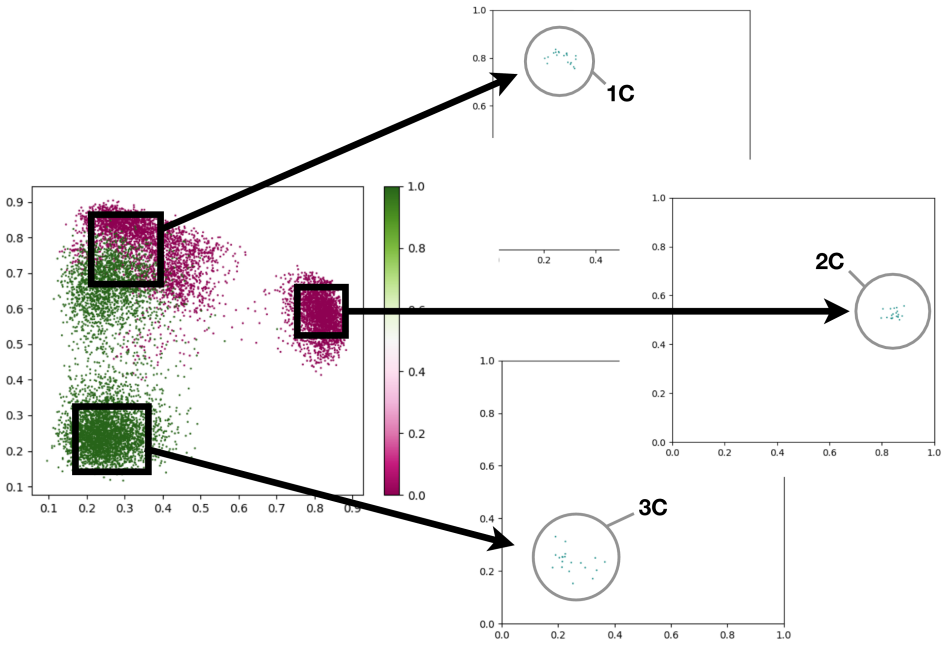
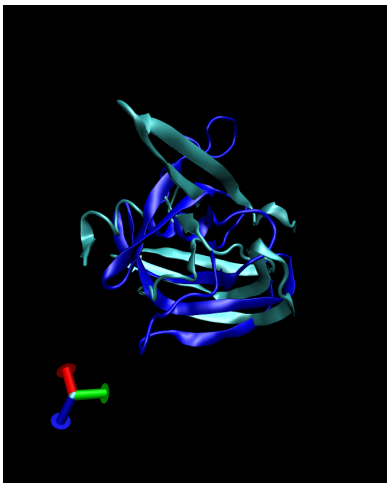



Figura 4.10: Estraggo un sample di distribuzioni di diedri da ognuno dei cluster che etichetto con 1-2-3C.



(a) Cluster 2C.

(b) Cluster 3C.

Figura 4.11: Viene mostrata in azzurro la conformazione spaziale corrispondente a una conformazione di diedri estratta dai due diversi cluster. In blu invece si ha la conformazione dello stato nativo. Le due conformazioni vengono quindi centrate tramite il calcolo dell'RMSD per averne un confronto valido.

estratto un set di conformazioni da ognuno dei tre cluster per studiarne la natura (Fig. 4.10).

Due corrispondono ancora allo stato nativo e denaturato, come si può osservare dal confronto delle configurazioni spaziali corrispondenti a due conformazioni di diedri estratte a caso dai cluster 2C e 3C (Fig. 4.11).

Estraggo invece delle distribuzioni di diedri prese dalla simulazione a temperatura più bassa (ossia alla temperature per cui la preoteina si trova nello stato nativo per un tempo considerevolmente più alto) dal cluster 1C. Si nota che tali distribuzioni corrispondono a configurazioni spaziali non molto differenti da quella dello stato nativo (Fig. 4.12): le ditribuzioni non sono quindi distinte dalla loro conformazione spaziale.

Si prova a vedere quindi se sussiste una correlazione tra i diedri (e quindi una dipendenza dalle strutture secondarie, individuate dai diedri come visto nei plot di Ramachandran) dello stato nativo con quelli di questa classe: dall' analisi fatta nella Fig. 4.13 non si nota alcun tipo di correlazione. La rete sta quindi dividendo le varie conformazioni in base alla formazione di alcune strutture secondarie o a certe disposizioni di diedri che noi non riusciamo ad individuare da questa prima analisi (contrariamente al caso di due soli stati in cui invece era chiaro fin da subito quali fossero le strutture cardine per la distinzione nelle due classi).

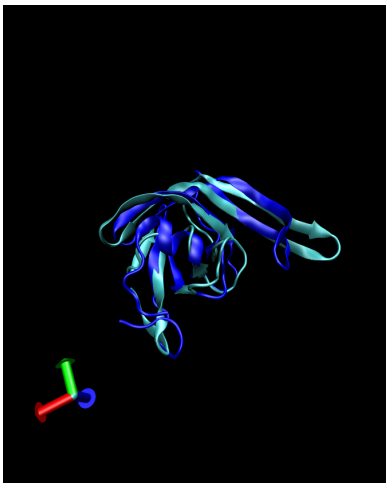


Figura 4.12: Come per i cluster 2C e 3C, viene mostrata la differenza tra le configurazioni spaziali corrispondenti allo stato nativo (blu) e a una distribuzione di diedri estratta dal cluster 1C (azzurro).

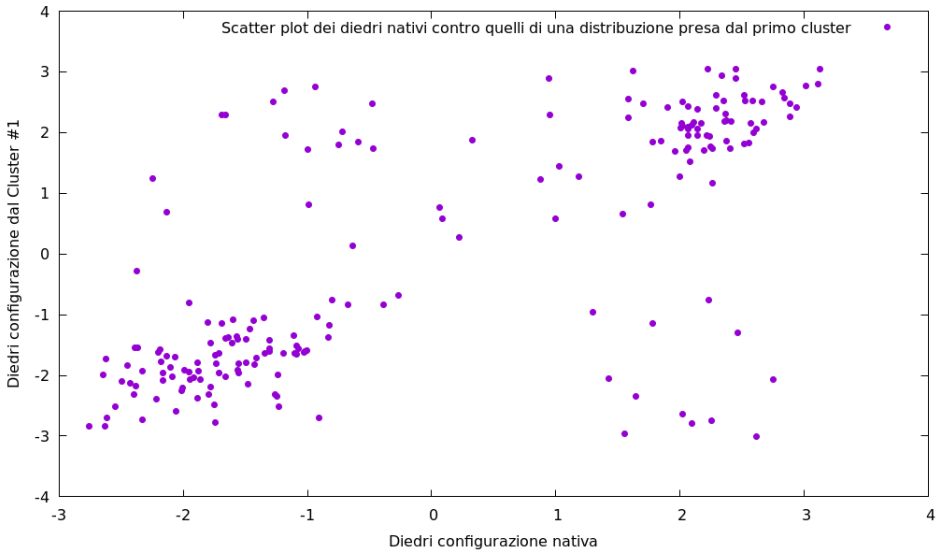



Figura 4.13: Grafico in cui vengono visualizzati i punti $(dih_i^{(nat)}, dih_i^{(1C)})$, dove $dih_i^{(nat)}$ sono i diedri della configurazione dello stato nativo e $dih_i^{(1C)}$ i diedri di una configurazione estratta dalla classe 1C.

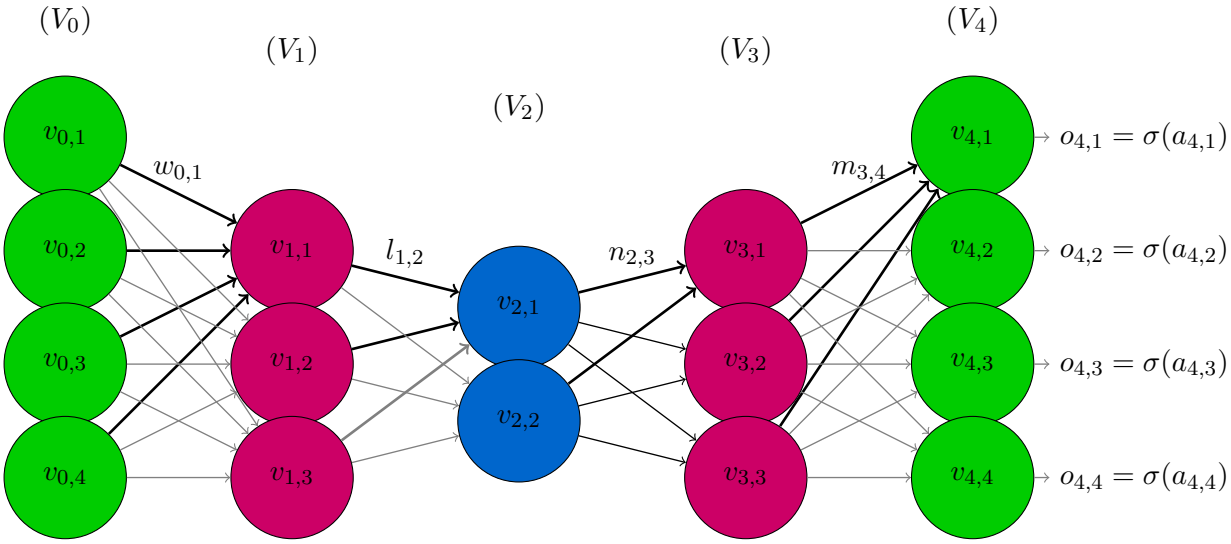



Figura 4.14: Esempio di autoencoder con hidden layer multipli. Nella parte sinistra della rete avviene la codifica dell'input nei nodi dello strato $V_2$. Nella parte destra invece si ha la decodifica, dove quindi il valore dei nodi in $V_2$ viene prima rimappato su $V_3$ e poi in output per riprodurre i valori dell'input.

## 4.5 Primo studio tramite metodi di deep-learning

Per l'ultima parte del mio lavoro si è utilizzata una rete autoencoder con una architettura più complicata con tre strati nascosti di dimensioni differenti, in cui quello centrale aveva ancora due soli nodi per poter studiarne le attivazioni (Fig. 4.14). Si utilizzano inoltre come funzioni di attivazione di tutti i nodi negli strati nascosti delle parrticolari funzioni chiamata *rectified linear units*, definite come

$$\sigma(a_{i,j}) = \begin{cases} m \cdot a_{i,j} & \text{se } a_{i,j} > 0 \\ 0 & \text{altrimenti} \end{cases}, \tag{4.1}$$

dove $m$ è un certo coefficiente angolare costumizzabile dall'utente.

Si sono quindi utilizzati ancora una volta i dati sulle mappe di contatto della traiettoria della proteina G per studiare la codifica dell'informazione nelle attivazioni dei due nodi dello strato interno come fatto per il caso dell'autoencoder a singolo hidden layer.

Visualizzando quindi le attivazioni su un grafico bidimensionale si nota come questa volta vengano identificate più di due classi (Fig. 4.15, sx). Per vedere di che classi si trattasse, abbiamo quindi visualizzato su un altro grafico le mappe di contatto decodificate a partire da paticolari valori dei due nodi interni (Fig. 4.15, dx).

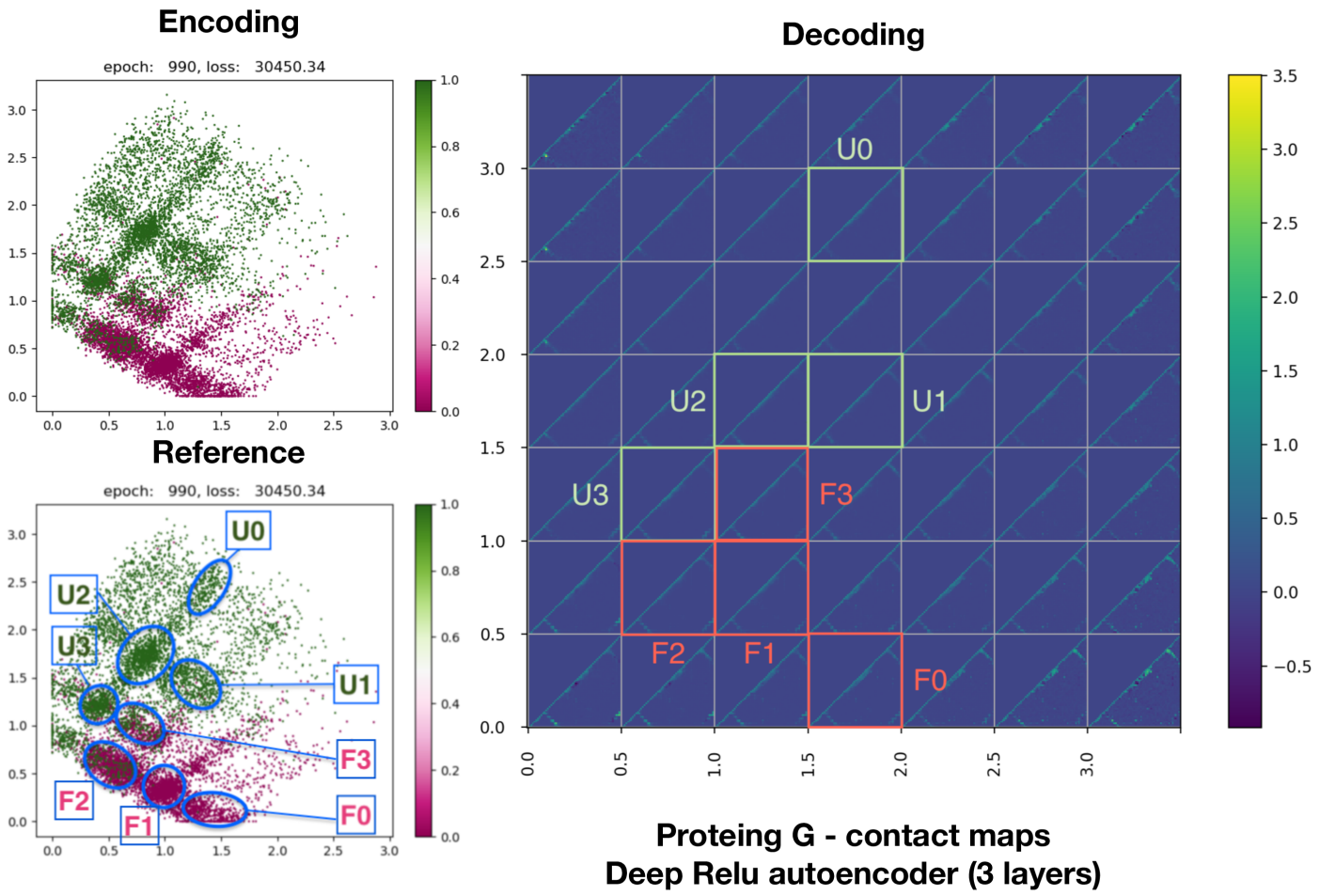
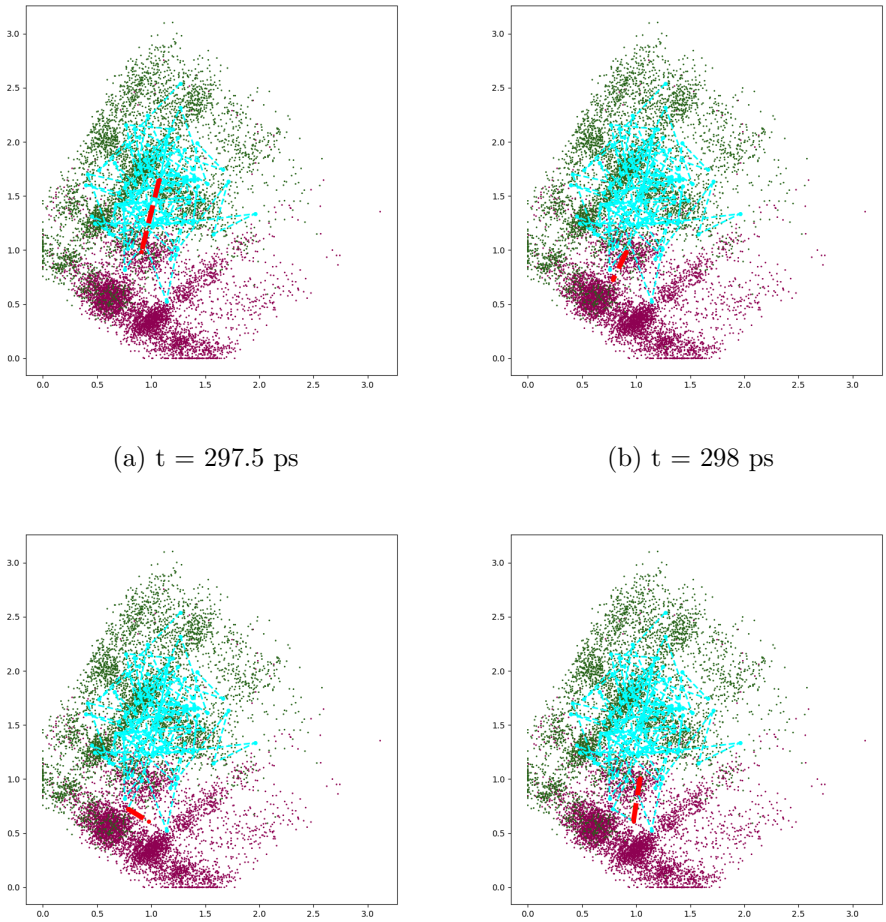



Figura 4.15: Confronto tra codifica (sinistra) e decodifica (destra) delle mappe di contatto della proteina G. In rosso le mappe corrispondenti allo stato nativo, mentre in verde quelle corrispondenti allo stato denaturato. Dal grafico della decodifica si osserva la formazione delle strutture secondarie andando dall'alto verso il basso, quindi il folding della proteina.



(a) t = 297.5 ps

(b) t = 298 ps

(c) t = 298.5 ps

(d) t = 299 ps

Figura 4.16: Snapshot a diversi tempi della traiettoria di folding. In rosso si ha il passo al dato tempo $t$, mentre in azzurro vengono visualizzati i passi precedenti.

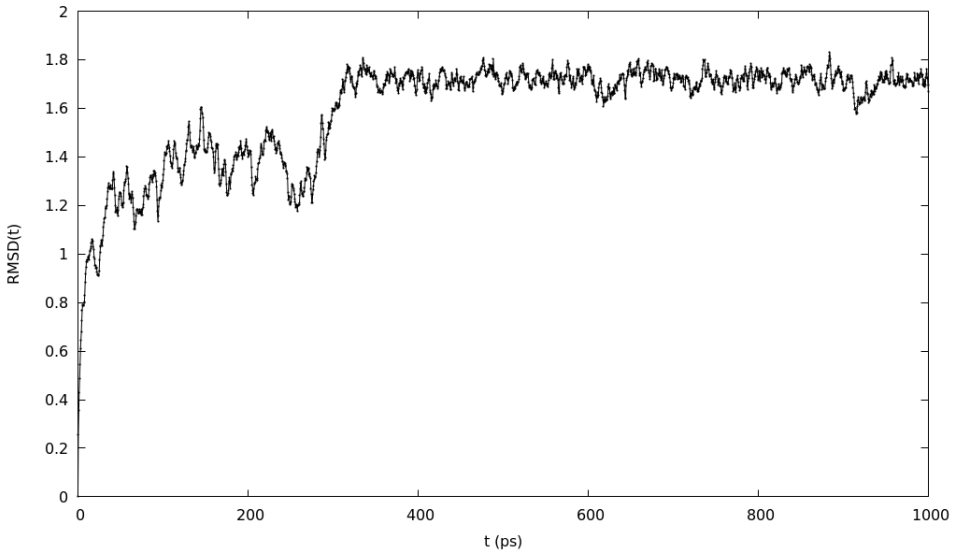



Figura 4.17: RMSD della proteina G con la conformazione dello stato nativo calcolato ad ogni passo della traiettoria lunga 1 ns.

Si nota quindi che le diverse mappe di contatto decodificate, potrebbero corrispondere a stadi intermedi della taiettoria di folding della proteina. Per avere una prima conferma della nostra ipotesi, abbiamo quindi provato a visualizzare una traiettoria di folding disegnando la codifica delle mappe di contatto corrispodenti ai diversi passi temporali della traiettoria (Fig. 4.16). In particolare, siamo andati ad utilizzare le mappe di contatto corrispondenti ai frame compresi in un certo intervallo di tempo entro il folding vero e proprio, che abbiamo identificato tramite il grafico dell'RMSD (Fig. 4.17). Ciò che si vede da questa prima analisi sembrerebbe confermare la nostra ipotesi: la proteina sembra che per passare da uno stato all'altro passi per il più delle volte da stati individuati dalle diverse classi (o che almeno si muova nella loro direzione).

# Conclusioni

Il lavoro svolto per questa tesi è stato quello di applicare gli algoritmi di reti neurali artificialial problema della riduzione dimensionale dei gradi di libertà conformazionali delle proteine. Dopo aver descritto le tecniche computazionali tipicamente usate in questo ambito e la raccolta dei dati nei capitoli 1 e 2, si presenta nei capitoli 3 e 4 il corposo lavoro di test che è stato eseguito.

Per quanto riguarda il lavoro di classificazione dello stato termodinamico delle proteine studiate, abbiamo mostrato che tutti i percettroni costruiti sono in grado di predire in modo soddisfacente lo stato delle proteine secondo i modelli utilizzati.

Anche lo studio delle traiettorie tramite gli autoencoder ha portato a risultati sorprendenti, soprattutto per quanto riguarda l'ultima parte di *deep learning* in cui, da una prima analisi, sembra che la rete riesca a classificare gli stati intermedi del folding di una proteina G. A proposito delle architetture a singolo strato nascosto, le reti utilizzate sono, nella metà dei casi, capaci di riprodurre la termodinamica del modello a due stati delle proteine studiate. Ciò rende le attivazioni dei nodi nello strato nascosto variabili ridotte utilizzabili per studi dei due sistemi all'equilibrio secondo il modello specificato. Ho parlato della metà dei casi poichè nell'altra metà la rete non divide le conformazioni in due soli classi (come si è visto per le distribuzioni dei diedri dell'HIV) o non divide proprio le conformazioni (come per le distribuzioni dei diedri della proteina G). Mentre le problematiche incontrate con le ditribuzioni dei diedri della proteina G sono probabilmente risolvibili tramite raffinazioni della rete utilizzata (ossia, tramite ottimizzazioni migliori dei parametri utilizzati o aumento della complessità della stessa), per quanto riguarda l'HIV proteasi un interessante prospettiva futura potrebbe essere riuscire a determinare se la terza classe distinta dalla rete corrisponda o meno a un vero stato termodinamico della proteina grazie ad analisi più complete, rispetto all'RMSD, del profilo di energia libera del sistema.

Si riconosce invece la necessità di uno studio più approfondito per quanto riguarda le modalità in cui reti di questo tipo riescono a codificare l'informazione fornitagli, per riuscire a capire, ad esempio, se vi sia o meno una corrispondenza tra il numero di nodi nello strato nascosto e il numero di stati del sistema studiato.

Subito a questa tesi seguirà uno studio più approfondito, qui solo ac-

cennato, delle traiettorie di folding tramite reti deep, metodo che, una volta dimostrata la sua efficacia su proteine la cui termodinamica è conosciuta (come la proteina G), potrebbe essere impiegato per scoprire nuove traiettorie di proteine la cui termodinamica è ancora sconosciuta.

# Ringraziamenti

Ringrazio innanzitutto tutto il gruppo del Laboratorio di Biofisica Teorica: Guido Tiana, Riccardo Capelli, Filippo Cola, Matteo Negri, Martina Crippa e Matteo Cagiada, coi quali ho avuto modo di discutere a proposito del mio problema. In particolare ringrazio Riccardo e Matteo N. per le spiegazioni dettagliate, la pazienza e la precisione nei suggerimenti, le soluzioni fornite e la grande disponibilità.

# Bibliografia


[1] Massimiliano Bonomi et al. "PLUMED: A portable plugin for free-energy calculations with molecular dynamics". In: *Computer Physics Communications* 180.10 (2009), pp. 1961–1972.

[2] Soumith Chintala. "An overview of deep learning frameworks and an introduction to pytorch". In: (2017).

[3] Nobuhiro Go. "Theoretical studies of protein folding". In: *Annual review of biophysics and bioengineering* 12.1 (1983), pp. 183–210.

[4] Ian Goodfellow et al. *Deep learning*. Vol. 1. MIT press Cambridge, 2016.

[5] Sachie Kimura et al. "The maturation of HIV-1 protease precursor studied by discrete molecular dynamics". In: *Proteins: Structure, Function, and Bioinformatics* 82.4 (2014), pp. 633–639.

[6] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[7] Derrick Nguyen. "Improving the learning speed of 2-layer neural networks by choosing initial values of the adaptive weights". In: *Proc. International Joint Conference on Neural Networks*. Vol. 3. 1990, pp. 21–26.

[8] Martin Riedmiller e Heinrich Braun. "A direct adaptive method for faster backpropagation learning: The RPROP algorithm". In: *Neural Networks, 1993., IEEE International Conference on*. IEEE. 1993, pp. 586–591.

[9] Hannes Risken. "Fokker-planck equation". In: *The Fokker-Planck Equation*. Springer, 1996, pp. 63–95.

[10] Shai Shalev-Shwartz e Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[11] LS SMITH. *INTRODUCTION TO THE THEORY OF NEURAL COMPUTATION-HERTZ, J, KROGH, A, PALMER, RG*. 1992.

[12] David Van Der Spoel et al. "GROMACS: fast, flexible, and free". In: *Journal of computational chemistry* 26.16 (2005), pp. 1701–1718.