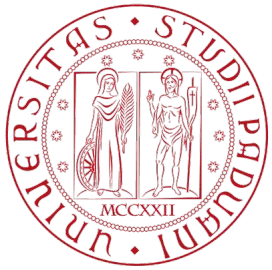


UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Laurea in Fisica

Tesi di Laurea

Studio della superficie potenziale della molecola d'acqua tramite machine learning

Relatore	Laureando
Prof./Dr. Paolo Umari	Alessio Tuscano

Anno Accademico 2023/2024

Studio della superficie potenziale della molecola d'acqua tramite machine learning

Introduzione

Lo scopo del presente lavoro è dimostrare come sia possibile addestrare un modello di rete neurale per simulare situazioni di fisica atomica ottenendo le proprietà generali di un sistema atomico generico.

In questo elaborato è stato fatto ampio uso del software libero Quantum Espresso [4] [3], in particolare della simulazione `pw.x` in configurazione *SCF* o *Self Consistent Field* in modo da calcolare le varie proprietà di un sistema atomico a partire dalle posizioni degli atomi.

Le simulazioni sono state effettuate in modo automatizzato tramite l'*API* Aiida [12] [8] e la libreria associata aiida-quantumespresso [14], questi hanno permesso l'automatizzazione del processo in modo efficiente ed efficace.

Tramite la tecnica a principi primi *DFT* sono stati quindi costruiti vari dataset contenenti informazioni di posizioni ed energia totale su un sistema a tre atomi: due atomi di idrogeno e uno di ossigeno. Per la costruzione è stata usata la libreria *pickle* in Python.

Questo set di dati è stato utilizzato poi per l'addestramento di modelli di rete supervisionati. La libreria utilizzata in questo caso è la libreria Tensorflow [1] per lo sviluppo di intelligenza artificiale ad alto livello. Dopo varie iterazioni si è ottenuta una rete con errore nell'ordine di 0.5 eV, ritenuto accettabile per lo scopo di questo elaborato e per l'ordine di grandezza delle predizioni richieste.

Tramite questo elaborato si mostra come è possibile addestrare un modello regressivo di predizione per prevedere varie proprietà da un sistema di molecole date alcune informazioni; in questo caso sono state usate le posizioni degli atomi.

Indice

Capitolo 1

Cenni di teoria

1.1 Richiami alla teoria DFT

La Teoria del funzionale densità, o *Density Functional Theory (DFT)*, è un metodo a principi primi per calcolare proprietà di un insieme di atomi.

Questo tipo di approccio si basa su una serie di assunzioni [13]:

- Approssimazione Born-Oppenheimer: nell'approssimazione Born-Oppenheimer si trattano soltanto gli elettroni come particelle quantistiche in quanto la velocità dei nuclei è considerevolmente minore di quella degli elettroni. Si separano quindi i potenziali di interazione con nuclei ed altri elettroni, inoltre gli elettroni rispondono istantaneamente alle sollecitazioni dei nuclei.
- Primo Teorema di Hohenberg-Kohn: data una densità elettronica allo stato fondamentale $n_0(\bar{r})$, il potenziale esterno è unicamente definito. Quindi la funzione d'onda ψ_0 può essere calcolata tramite la risoluzione dell'equazione di Schrödinger e di conseguenza si può determinare l'energia totale $E[n(\bar{r})]$.
- Secondo Teorema di Hohenberg-Kohn: la densità elettronica che minimizza l'energia è esattamente la densità elettronica fondamentale $n_0(\bar{r})$ e l'energia corrispondente è E_0. Dal principio variazionale vale inoltre che $E_0 \leq E[n(\bar{r})]$.
- Ipotesi di Kohn-Sham: approssimando ulteriormente possiamo dire che gli elettroni nel sistema non interagiscono tra di loro previa la scelta di un potenziale efficace $v_s(\bar{r}_i)$ che rende la densità elettronica del sistema di elettroni non-interagenti uguale a quella del sistema di elettroni interagenti tra loro. Le funzioni d'onda del sistema non-interagente sono denominate di Kohn-Sham e l'equazione da risolvere è:

$$\hat{H}_s \phi_i(\bar{r}) = \left[-\frac{\hbar^2}{2m_e} \sum_{i=1}^{N} \nabla_i^2 + \sum_{i=1}^{N} v_s(\bar{r}_i) \right] \tag{1.1}$$

1.1.1 Derivazione Hamiltoniano di Kohn-Sham

Tramite l'approccio di Kohn-Sham il problema è quindi cambiato dal trovare la densità elettronica allo stato fondamentale $n_0(\bar{r})$ a trovare le funzioni d'onda di Kohn-Sham. Possiamo quindi scrivere il funzionale energia totale:

$$E[n(\bar{r})] = T_s[n(\bar{r})] + E_H[n(\bar{r})] + E_{xc}[n(\bar{r})] + \int v_{ext} n(\bar{r}) \tag{1.2}$$

dove T_s è l'energia cinetica, E_H è l'energia di Hartree del sistema non-interagente, E_{xc} è il funzionale di scambio e correlazione definito come $E_{xc}[n(\bar{r})] \equiv T[n(\bar{r})] - T_s[n(\bar{r})] + V_{ee}[n(\bar{r})] - E_H[n(\bar{r})]$; dove V_{ee} è il potenziale quantistico d'interazione elettrone-elettrone.

Derivando il funzionale energia rispetto alla densità elettronica si può ottenere l'hamiltoniano corrispondente:

$$\hat{H}_s = \frac{\delta E[n(\bar{r})]}{\delta[n(\bar{r})]} = -\frac{\hbar^2}{2m_e}\sum_{i=1}^{N}\nabla_i^2 + v_{ext}(\bar{r}) + \int \frac{n(r')}{|\bar{r}-\bar{r}'|}dr' + v_{xc}(\bar{r}) \tag{1.3}$$

In questo caso $\hat{H}_s$ dipende direttamente da $[n(\bar{r})]$ attraverso il termine di scambio-correlazione $v_{xc}[n(\bar{r})]$ e il termine integrale, ovvero il potenziale di Hartree, che approssima l'interazione elettrone-elettrone attraverso un campo medio integrato sullo spazio.

Inoltre la densità elettronica $[n(\bar{r})]$ dipende a sua volta dalle soluzioni ϕ_i dell'equazione 1.1 , in quanto:

$$n(\bar{r}) = \sum_{i=1}^{N} |\phi_i(\bar{r})|^2 \tag{1.4}$$

Il tutto necessita quindi di un approccio iterativo per la risoluzione [13].

1.1.2 Funzionali utilizzati

Per i calcoli necessari all'elaborato sono quindi stati scelti dei funzionali di scambio correlazione di tipo *LDA*, ovvero in *Local Density Approximation*. In questa approssimazione il funzionale energia è una funzione della densità locale:

$$E_{xc} = \int n(\bar{r})\epsilon_{xc}(n(\bar{r}))d\bar{r} \tag{1.5}$$

$$v_{xc}(\bar{r}) = \epsilon_{xc}(n(\bar{r})) + (n(\bar{r}))\frac{d\epsilon_{xc}(n)}{dn} \tag{1.6}$$

dove $\epsilon_{xc}(n)$ viene ottenuto tramite tecniche Monte Carlo quantistiche. Questa approssimazione può risultare in sovrastime dell'energia causate dalle maggiori interazioni tra sistemi e lunghezze di legame più corte [13].

1.2 Richiami alle reti neurali

Le reti neurali sono strutture algoritmiche ispirate al cervello. Ogni rete neurale è composta da diversi strati, chiamati *layers*, ed ogni strato è formato da un numero fissato di neuroni.

I neuroni sono quindi l'unità computazionale di base di una rete neurale. Sono descritti da una funzione di attivazione $\sigma(z)$ variabile in base al tipo di neurone utilizzato. Il parametro z dipende dalle *features* x_j e da pesi ω_j associati ad ogni x_j, attraverso la relazione:

$$z = \sum_{i=0}^{N} \omega_j x_j \tag{1.7}$$

Tutti i neuroni hanno una proprietà fondamentale, ovvero i pesi ω_j associati alle *features* delle classi di addestramento. Questi infatti determinano l'attivazione del neurone in base all'input dato e sono quindi responsabili della direzione della rete in base ad uno specifico input.

Un'altra proprietà fondamentale del neurone è la sua funzione di attivazione: un esempio ne è la funzione di attivazione del *perceptron*, che si può considerare come l'antenato del neurone artificiale moderno. La sua funzione di attivazione è la funzione gradino:

$$\tilde{\sigma}(z) = \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases} \tag{1.8}$$

Il perceptron si comporta bene con problemi di classificazione separabili, tuttavia le prestazioni per problemi non separabili lasciano a desiderare. Il neurone artificiale è un'evoluzione del *perceptron* in quanto permette di usare diverse funzioni di attivazione in base al problema da risolvere.

1.2.1 Funzioni di attivazione dei neuroni artificiali

Per il problema affrontato sono stati utilizzati neuroni con funzioni d'attivazione differenti:

- Funzione d'attivazione $ReLU$:

$$\sigma(z) = \max(0, z) = \begin{cases} z & z > 0 \\ 0 & altrimenti \end{cases} \tag{1.9}$$

  la funzione d'attivazione *Rectified Linear Unit* ($ReLU$) è una funzione con diverse proprietà:

  - ha un basso costo computazionale, simile all'addizione o alla moltiplicazione [6]
  - è invariante rispetto a costanti
  - ha un comportamento quasi-lineare
  - hanno buone prestazioni in modelli multi-strato con apprendimento non supervisionato [5]

- Funzione di attivazione $GELU$:

$$\text{GELU}\,(x) = xP\,(X \leq x) = x\Phi\,(x) = x \cdot \frac{1}{2}\left[1 + \text{erf}(x/\sqrt{2})\right] \tag{1.10}$$

  La funzione d'attivazione *Gaussian Error Linear Unit* ($GELU$) è una variante moderna della più comune $ReLU$. E' stata introdotta nel 2016 da Dan Hendrycks e Kevin Gimpel per far fronte ad alcuni problemi della $ReLU$, come la sua non differenziabilità in 0.

  La $GELU$ infatti è una versione più morbida della $ReLU$, pur mantenendone in gran parte i benefici [7].

Questi diversi tipi di neuroni sono stati utilizzati per una rete di tipo *DNN*, ovvero una *Deep Neural Network*.

Le *Deep Neural Network* sono uno dei modelli basilari di reti neurali: infatti consistono di un *layer* di input con dimensione pari al numero di *features*, un *layer* di output con la dimensione pari al numero di *labels* e un numero non definito di *layers* tra i due citati sopra, chiamato comunemente *black box*.

Le *features* sono il numero di parametri per campione associati ad un singolo *label*, in sostanza sono i parametri che verranno dati in input alla rete già addestrata, che dovrà prevedere un risultato il quanto più vicino possibile ad un *label*, ovvero l'insieme di parametri che la rete deve prevedere.

I *layers* sono i cosiddetti strati della rete, ovvero un insieme di neuroni che generalmente non possono creare collegamenti tra essi, ma solo con altri neuroni dello strato precedente e/o successivo.

1.2.2 Tipi principali di reti neurali

La rete neurale di tipo *DNN* è una rete di tipo *feed-forward*, ovvero la cui informazione si muove solo in un unico verso. Come suggerisce il nome stesso, le reti di tipo *feed-forward* fanno fluire l'informazione solo in avanti; il tipo più semplice di rete *feed-forward* è la *FNN* o *Feedforward Neural Network*, ovvero una rete con un singolo *layer* intermedio tra quello di input e quello di output.

Le *DNN* sono ormai lo standard per molti problemi e differiscono semplicemente dalle *FNN* per il numero di strati nascosti, o *hidden layers*, tra il primo e l'ultimo strato.

Ci sono altri tipi di reti neurali, come le *Recurrent Neural Networks* che fanno fluire l'informazione in entrambi i versi, o le *Convolutional Neural Networks* che sono formate da strati di tipo convoluzionale progettati appunto per problemi di *Computer Vision*. Per lo scopo di questo elaborato è stata scelta una *DNN* in quanto il problema è di tipo regressivo [6].

1.2.3 Addestramento supervisionato

Per affrontare il problema in modo da ridurre il numero di campioni necessari e aumentare considerevolmente l'accuratezza e la precisione del modello si è adottato un addestramento di tipo supervisionato.

I due tipi principali di addestramento sono l'addestramento supervisionato e l'addestramento non supervisionato:

- Addestramento non supervisionato: questo tipo di addestramento viene principalmente utilizzato per riordinare o catalogare i dati di addestramento. Un altro uso di questo tipo di addestramento è anche il *clustering*, ovvero la divisione dell'insieme di dati in sottoinsiemi da esempi simili.
- Addestramento supervisionato: è il tipo di addestramento che permette di risolvere problemi come quelli di classificazione o di regressione. La differenza principale con l'addestramento non supervisionato è che il dataset di addestramento viene suddiviso in due macrocategorie come spiegato prima. Ad ogni *feature*, o esempio, viene associato un *label*, o bersaglio. La rete ha quindi la possibilità di imparare ad ogni iterazione, o *epoch*, le proprietà delle *features* e confrontarle con i *labels*.

Il termine "addestramento supervisionato" viene dal concetto che alla rete viene già fornita la risposta al problema da risolvere similmente a come un docente può già fornire la soluzione ad un quesito che pone ai suoi studenti [6].

In aggiunta a questo controllo il dataset è stato suddiviso ulteriormente in tre parti:

1. *Training set*: l'insieme di addestramento è direttamente coinvolto all'inizio dell'addestramento, la rete impara direttamente da questo insieme ed i pesi vengono stabiliti attraverso i *samples* contenuti in questo insieme.
2. *Validation set*: l'insieme di validazione viene utilizzato alla fine di ogni epoca per convalidare i pesi modificati. Contribuisce all'accuratezza e precisione nel modello, nonchè in modo significativo all'efficienza dell'addestramento.

3. *Test set*: l'insieme di test viene utilizzato alla fine dell'addestramento per misurare le prestazioni del modello addestrato.

La presenza di ulteriori insiemi rispetto a quello d'addestramento non determina la distinzione in addestramento supervisionato o meno, però gli insiemi di validazione e di test contribuiscono certamente all'efficienza in fase di ricerca e addestramento di un modello ottimizzato per un problema specifico [6].

1.2.4 Regolarizzazione L^2

Ad ogni strato, escluso quello di output, sono stati aggiunti i *kernel reguralizers* di tipo L^2. Questa è una diretta conseguenza del *No Free Lunch Theorem*: teorema che dice che non esiste un modello che abbia prestazioni migliori di qualsiasi altro in senso universale, ovvero tutti i modelli hanno la stessa possibilità d'errore su punti non precedentemente classificati data una distribuzione media di tutti i possibili punti generati [6].

Quindi il teorema implica che è necessario cucire il modello in modo da avere buone prestazioni sul problema specifico. La regolarizzazione è in generale una qualsiasi modifica fatta ad un algoritmo di addestramento atta a ridurre l'errore di generalizzazione ma non il suo errore in addestramento.

La regolarizzazione utilizzata in questo modello è di tipo L^2, ed è conosciuto principalmente come *weight decay*. Questa strategia penalizza maggiormente i pesi agli estremi aggiungendo il termine $\Omega(\theta) = \frac{\lambda}{2}\|\omega\|_2^2$, ed è conosciuta anche come regolarizzazione *Ridge*. λ è il parametro di regolarizzazione che regola la percentuale di regolarizzazione nel computo del peso w_j; in seguito verrà chiamato `l2par`.

Questo tipo di regolarizzazione è molto utile per prevenire *overfitting* nel modello: ovvero quel fenomeno per cui la rete si adatta troppo al set di addestramento, con conseguenti cattive prestazioni nel momento in cui si chiede una predizione con valori differenti da quelli nel suddetto insieme [6].

Capitolo 2

Implementazione

2.1 Software utilizzato per simulazioni *SCF*

Lo scopo di questo elaborato richiedeva un software fondamentale per simulazioni *SCF*, o *Self Consistent Field* adottando la teoria a principi primi *DFT* per calcolare le proprietà di un aggregato di atomi.

L'obiettivo in questa parte è di ottenere un insieme di *samples*, o un *dataset*, pronto per poter essere utilizzato come *dataset* di addestramento per la rete neurale.

Questi sono i passaggi chiave eseguiti per generare un insieme pronto per essere importato nel programma di addestramento della rete neurale, e successivamente suddiviso in *training set*, *validation set* e *test set*:

- Simulazione con immissione manuale tramite `pw.x` e `water.in` da terminale, con uscita `water.out`
- Implementazione della medesima simulazione su un programma Python attraverso l'*API* `Aiida` e la libreria `aiida-quantumespresso`
- Modifica del programma per incorporare la generazione casuale delle posizioni degli atomi nella simulazione
- Implementazione di un ciclo per effettuare n iterazioni e scrivere i risultati all'interno di un file `dataset.pickle`

Di seguito verranno illustrati i passaggi su citati.

2.1.1 Simulazione di una molecola di H_2O

E' stato utilizzato il software Quantum Espresso, in particolare il codice `pw.x` al suo interno.

Il codice `pw.x` richiede un file di input `filename.in` dove ci sono scritte le informazioni del sistema da analizzare, in particolare il tipo di simulazione da effettuare, *SCF* nel nostro caso, i percorsi degli pseudopotenziali, di tipo *LDA* nel nostro caso, e altri parametri prettamente pertinenti alla simulazione come soglie di accuratezza o tecniche di minimizzazione.

Il codice è eseguibile manualmente da riga di comando e può o meno restituire un file output `filename.out` con all'interno i risultati della simulazione, ovvero le varie proprietà della molecola, forze, energie, numero di orbitali ecc.

Per interfacciarsi in modo efficace con il codice `pw.x`, data la mole di simulazioni, è stata scelta l'*API* `Aiida` associata alla libreria `aiida-quantumespresso`. Questo permette di poter effettuare chiamate al codice `pw.x` in modo automatico, tenendo separata la computazione numerica dalla gestione delle

chiamate al codice `pw.x`. Permettendo al contempo la gestione e l'eventuale manipolazione dei risultati ottenuti.

In questo caso `Aiida` è stato implementato nella stessa macchina in cui risiede il codice `pw.x` compilato. Questa soluzione, nonostante non fosse la più pratica ed efficiente dal punto di vista computazionale, è stata adottata per la necessità di ottenere un ambiente di lavoro funzionale nel minor tempo possibile, data la quantità di simulazioni richieste per un problema regressivo di questo tipo.

2.1.2 Generalizzazione e costruzione del dataset

Il passo successivo alla simulazione di una molecola d'acqua in configurazione naturale è stato di impostare la generazione di una cella con coordinate spaziali degli atomi casuali, per ottenere un campione di aggregato di due atomi di idrogeno e uno di ossigeno casuale. L'idea base è quella di sfruttare questa generazione casuale ed ottenere un insieme di dati variegato e pronto per essere usato direttamente per l'addestramento della rete neurale.

Si è proceduto con una generazione casuale uniforme attraverso la funzione `random.uniform(min,max)` appartenente alla libreria `numpy` per generare le coordinate cartesiane relative alle posizioni degli atomi. La matrice posizioni viene quindi utilizzata per costruire una cella cubica con caratteristiche fissate. Per ovviare a problemi di minimizzazione e convergenza relativi al conto *SCF* si è optato per aggiungere uno *smearing* gaussiano alle opzioni di `pw.x`. Questa impostazione permette tolleranze maggiori per le soglie dell'energia degli orbitali elettronici, il che favorisce la convergenza della simulazione nel complesso.

Inoltre per limitare gli effetti di bordo, dato che si studia la molecola isolata, è stato limitato l'effettivo volume di generazione degli atomi al cubo della metà del lato della cella di simulazione.

La dimensione scelta per la cella di simulazione è di $L_s = 10$ Å; all'interno di questo cubo si trova l'effettiva regione cubica di lato $L_e = \frac{L_s}{2} = 5$ Å.

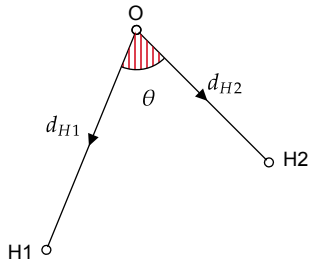
Successivamente si è deciso di ridurre la complessità della rete trasformando le coordinate da cartesiane a simmetrizzate. Per questo sistema si è scelto l'atomo di ossigeno come centro delle coordinate, poi si è calcolata la distanza tra quest'ultimo e i due atomi di idrogeno, infine l'angolo tra i due atomi di idrogeno. Si è quindi ridotto il grado di complessità di parametri da dare in input alla rete da 9 parametri, ovvero le coordinate cartesiane, a soli 3 parametri sfruttando la simmetria del sistema a tre corpi:

$$\begin{pmatrix} d_{H_1} \\ d_{H_2} \\ \theta \end{pmatrix} \longleftrightarrow \begin{pmatrix} H_{1x} & H_{1y} & H_{1z} \\ H_{2x} & H_{2y} & H_{2z} \\ O_x & O_y & O_z \end{pmatrix} \tag{2.1}$$

Questo cambio di sistema di riferimento è stato attuato tramite una funzione apposita `desymmetrize3p` responsabile della trasformazione di simmetria T, dove viene scelto arbitrariamente

il piano xy per proiettare le posizioni, si ha quindi che:

$$T = \begin{cases} H_{1x} = O_x + d_{H1} \sin\frac{\theta}{2} \\ H_{1y} = O_y + d_{H1} \cos\frac{\theta}{2} \\ H_{2x} = O_x - d_{H2} \sin\frac{\theta}{2} \\ H_{2y} = O_y + d_{H2} \cos\frac{\theta}{2} \\ H_{nz} = O_{x,y,z} = 5.0\text{Å} \end{cases} \tag{2.2}$$



Simmetria utilizzata

Dove nelle coordinate della cella l'atomo di ossigeno si trova al centro e gli atomi di idrogeno variano le loro coordinate planari secondo la trasformazione di simmetria T.

Le coordinate generate casualmente sono quindi nel sistema di riferimento simmetrizzato, e vengono salvate così generate associate all'energia della superficie potenziale del sistema a conto *SCF* finito. Vengono poi trasformate in coordinate cartesiane per generare la struttura atomica che verrà poi utilizzata dal codice `pw.x` assieme ai parametri di configurazione per effettuare la simulazione.

Successivamente il codice è stato incapsulato in un ciclo per automatizzare il processo, dove ad ogni iterazione sono stati salvati i parametri di uscita interessati, ovvero l'energia totale del sistema, e le posizioni simmetrizzate degli atomi in due liste.

Si è impostato un parametro di convergenza $conv_{thr} = 10^{-6}$ in modo da rendere l'errore delle simulazioni trascurabile rispetto all'errore in apprendimento, pur mantenendo una buona velocità di calcolo.

Le liste generate sono state utilizzate infine per costruire il *dataset* con le posizioni associate alla parola chiave `positions` e le energie prese dall'output del nodo relativo con parola chiave associata `energy`.

Per ulteriori dettagli si rimanda al codice integrale in appendice B chiamato `'pwbase_workflow_datasetcreation_aiida.ipynb'`.

2.2 Addestramento della rete neurale

Con il *dataset* pronto è stato possibile addestrare la rete neurale, anche in questo caso sono stati effettuati diversi passaggi prima di arrivare ad una rete con buone prestazioni:

- Importazione e divisione del *dataset* nei tre set di *Training set*, *Validation set*, *Test set*
- Costruzione del modello di rete neurale con i relativi parametri
- Addestramento e test sul *Test set*
- Test aggiuntivi su campioni limitati di situazioni fisiche interessanti

2.2.1 Costruzione del modello di rete

Come accennato in precedenza è stata utilizzata l'*API* `Tensorflow` [1] con complementi le librerie seguenti:

1. `pickle` per la creazione, esportazione ed importazione dei *dataset*: la libreria è stata utilizzata sia durante la fase di creazione, come descritto precedentemente, sia durante l'importazione; infatti attraverso l'associazione per parole chiave, similmente ad una mappa, è stato possibile dividere il dataset in *features* e *labels*.

2. `scikit-learn` per la suddivisione nei set di *Training set*, *Validation set*, *Test set*: le posizioni e le energie divise rispettivamente in *features* e *labels* sono poi state ulteriormente divise attraverso la funzione `sklearn.model_selection.train_test_split` chiamata due volte con parametro `test_size=0.01` e `test_size=0.1`. Attraverso queste due chiamate si suddividono quindi i campioni come segue rispetto al totale.

Train set	0.90
Validation set	0.09
Test set	0.01

Tabella 2.1: Suddivisione del dataset creato per l'addestramento

3. `keras` per la costruzione e gestione del modello di rete neurale: per costruire il modello è stata definita una funzione `createmodel`, questa riceve come input la classe di input `input_class`, la classe di output `output_class`, il parametro `base_neurons` e il parametro `l2par`. Questi tre parametri possono variare: la classe di input e di output sono rimaste sempre rispettivamente 3 e 1, ovvero le dimensioni delle posizioni e l'energia totale; il parametro `base_neurons` invece è stato variato attraverso diverse iterazioni ed i vari strati hanno un numero di neuroni calcolato in base a questo parametro. Nello specifico ad ogni strato viene assegnato un numero di neuroni pari a metà dello strato precedente. Il parametro `l2par` invece è il peso relativo alla regolarizzazione L^2.

   Come detto precedentemente per ogni strato ad eccezione di quello finale sono stati utilizzate regolarizzazioni attraverso il parametro `kernel_reguralizer=(l2par)`. Il modello di tipo sequenziale utilizza sia strati con funzione di attivazione *GELU* che *ReLU*.

Successivamente un'altra funzione definita come `compilemodel` si prende cura di compilare il modello, con ottimizzatore `opt` dato in chiamata, funzione perdita `loss='mean_squared_error'` e come metrica aggiuntiva all'errore quadratico medio è stato scelto l'errore medio assoluto.

Queste due metriche, assieme alla suddivisione del *dataset*, hanno permesso di valutare la qualità dell'addestramento immediatamente senza la necessità di test aggiuntivi su ulteriori campioni strettamente casuali.

2.2.2 Addestramento del modello di rete

Con il modello creato e i set di addestramento pronti per essere utilizzati si è potuto effettuare l'addestramento, in questa fase si è impostato un ciclo di iterazione per trovare il miglior modello tra una serie di iterazioni in cui si sono variati i seguenti parametri:

- `batch_size`: numero di divisioni del *Training set*, impatta la velocità di apprendimento e la qualità di quest'ultimo.

- `epochs`: questo è il parametro che influenza maggiormente l'accuratezza e la precisione del modello dopo aver fissato il design della rete, può essere motivo di *underfitting* o *overfitting*.
- `base_neurons`: numero di neuroni dello strato iniziale, ad ogni strato successivo il numero di neuroni si dimezza nel caso di design a cono.
- ottimizzatore: sono stati testati gli ottimizzatori `Adam, RMSprop, Adadelta, Adagrad` e altri incorporati nella libreria `Keras`. Si è scelto di utilizzare `Adam` durante la fase di ricerca del modello data la sua velocità di convergenza iniziale, e infine `Adadelta` per l'addestramento finale per raggiungere un minimo della *loss function* minore.
- numero di strati: è stato variato tra 4 e 7 strati con diverse configurazioni e si sono ottenuti i migliori risultati con 6 strati compreso quello finale.
- tipi di funzioni di attivazione: sono state provate reti con funzioni di attivazione esclusivamente *ReLU*, *GELU*, e sigmoidi. La funzione di attivazione è stata variata dopo aver definito il `batch_size`, e il parametro `base_neurons`.
- design della rete: sono stati provati i design a cono, cono inverso, con dimezzamento o raddoppiamento dei neuroni e con numero di neuroni variabile per iterazione. Si è scelto il design a cono diretto con dimezzamento del numero di neuroni per strato.
- parametro di regolarizzazione $\lambda =$ `l2par` per ogni strato: questo parametro è stato variato alla fine del processo di ricerca del modello; è stato trovato un $\lambda = 0.001$ che permette un addestramento efficace prevenendo al contempo *overfitting*.
- `callback`: la funzione `keras.callbacks.EarlyStopping` ha permesso di bloccare l'addestramento nel caso la `loss` non migliorasse entro un numero di epoche fissato a `pratience` $= 150$ prevenendo *overfitting*.

Sono state incorporate nel ciclo, dopo l'addestramento effettivo, delle procedure per facilitare la valutazione di ogni modello su diversi campioni rappresentativi di situazioni particolari, come formazione dell'idrogeno molecolare H_2 e formazione del gruppo ossidrilico OH. Di ogni modello generato sono stati salvati i pesi e i grafici relativi sia alla *loss function* che alle situazioni fisiche che verranno descritte.

Il *dataset* di addestramento è formato da campioni di atomi generati casualmente con distribuzione uniforme su tutta la cella effettiva scelta di lato $L_e = 5$ Å, questa dimensione ha permesso lo studio della formazione dei legami tramite *DFT* e la riproduzione tramite rete neurale delle seguenti casistiche:

1. $0 < \theta < \pi$ e $d_{H1} = d_{H2} = 0.96$ Å: questa è una configurazione di controllo per verificare le prestazioni della rete. Le distanze sono fisse a $d = 0.96$ Å, corrispondente alla distanza tipica del legame con l'atomo d'ossigeno in configurazione H_2O; in questo modo si può notare il variare del potenziale rispetto al parametro θ. Si potrebbe fare la stessa cosa tenendo fisso l'angolo e una distanza variando la terza, tuttavia si è notato come θ sia il parametro più difficile da prevedere tra i 3 e si è scelto quest'ultimo come parametro di controllo su campioni casuali.
2. $0 < \theta < \frac{\pi}{2}$ e $d_{H1} = d_{H2} = 2$ Å: questa configurazione permette di simulare la formazione dell'idrogeno molecolare H_2 tenendo fissa la distanza dall'ossigeno e variando l'angolo tra gli atomi di idrogeno in modo da osservare la formazione del legame e la variazione del potenziale superficiale in funzione di essa.
3. $0.1 < d_{H1} < 2$Å, $\theta = 1.82$ rad e $d_{H2} = 3$ Å: in questa configurazione invece si tiene a distanza un atomo di idrogeno e si simula la formazione del legame del gruppo idrossilico OH.

4. $0.5 < d_{H1} < 3.5$ Å, $\theta = 0.37$ rad e $d_{H2} = 2$ Å: in questa configurazione si utilizza il $\theta_{min} = 0.37rad$ ottenuto dall'analisi precedente per simulare la formazione di entrambi i legami al variare della distanza di un atomo di idrogeno. Questa è la configurazione più complicata affrontata in questo studio.

Di seguito al grafico 2.1, verrà mostrata la rete che rappresenta al meglio i diversi scenari elencati sopra [11]:

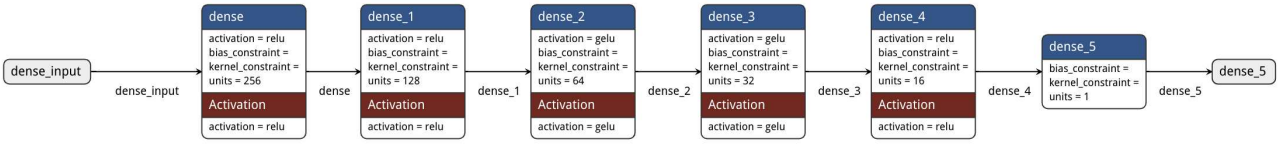


FIGURA 2.1: Schema della rete neurale finale utilizzata

La rete neurale scelta è schematizzata qui sopra: come accennato prima i design è a cono, il primo strato è formato da un numero di neuroni dato dal parametro `neurons` assegnato alla funzione che si occupa di inizializzare il modello. In input viene dato quindi il vettore contenente le coordinate (d_{H1}, d_{H2}, θ) a ciascuno dei 256 neuroni, mentre in output uscirà solo il parametro dell'energia da confrontare poi con il *label* `energy`. Gli strati successivi sono composti da un numero di neuroni dimezzato rispetto allo strato precedente, con due dei 5 strati che si occupano di effettuare la regressione con funzione di attivazione $GELU$, quelli con 64 e 32 neuroni, i restanti 3 invece utilizzano tutti la funzione di attivazione $ReLU$. Per ulteriori dettagli si rimanda al codice in appendice B chiamato `'water_dnn.ipynb'`.

Si procede ora con l'analisi delle diverse casistiche enunciate sopra:

1. Il parametro più complesso da imitare per la rete è il parametro θ e come si vede dal grafici A.2 e A.1 la rete neurale prevede molto bene la curva del potenziale per d_{H2} libero, e meno bene quella per θ libero.

   Ricordando che l'errore quadratico medio ottenuto in addestramento è dell'ordine di $\sigma = 0.2eV$ come si vede nel grafico 2.2 e osservando la scala di energia dei due grafici si nota come le predizioni in generale siano in linea con l'errore associato alla rete.

   In particolare per il grafico relativo al parametro libero d_{H2} si nota come la curva relativa alle predizioni della rete neurale in blu approssimi molto bene la curva arancione relativa al potenziale calcolato tramite DFT, vi è un leggero scostamento rispetto alla coda del controllo. Osservando invece il grafico relativo al parametro libero θ si nota come l'andamento delle due curve sia il medesimo ma la curva delle predizioni si discosti maggiormente dal controllo nella regione del minimo e nella coda.

   Questo suggerisce la necessità di migliorare la parte di addestramento e l'unica cosa possibile senza andare in *underfitting* oppure *overfitting* è aumentare la dimensione del *dataset* di addestramento. Purtroppo questa opzione non è pratica con le risorse a disposizione dato che la generazione di nuovi campioni da aggiungere al *dataset* avrebbe allungato enormemente i tempi con le risorse computazionali a disposizione. Per generare i circa 25000 campioni complessivi utilizzati tramite il ciclo *SCF* descritto sopra ci sono volute circa 300 ore di computazione con l'hardware[1] e il software[2] a disposizione. Si ricorda inoltre che l'errore associato alla rete non scala linearmente

[1] Intel Core i7 8700k con istruzioni AVX2
[2] Intel OneAPI e Intel MPI in Intel HPC Toolkit

con la quantità di campioni disponibili per l'addestramento a parità di modello scelto; si pensa siano quindi necessari ulteriori risorse computazionali che vanno oltre lo scopo di questo elaborato.

Ritornando ai grafici relativi ai parametri liberi si può notare come la scelta di fissare i due rimanenti nei valori di configurazione abbia permesso di trovare il valore minimo delle predizioni e confrontarlo con il minimo relativo alle curve di controllo.

$$\theta_{minDNN} = \theta_{minDFT} = 0.37rad \qquad d_{H2minDNN} = d_{H2minDFT} = 0.96\text{Å} \quad E = -480.4eV$$

Nel caso di θ il minimo si discosta di circa 0.05 rad e il suo valore energetico corrispondente è entro un $\sigma = 0.2eV$, mentre nel caso di d_{H2} il minimo combacia con il minimo della curva di controllo e questi a loro volta combaciano col il valore noto di lunghezza del legame dell'acqua $d_{Hth} = 0.96$ Å. In entrambi i casi si nota come il minimo dei due potenziali corrisponda a circa $E = -480.4eV$, ovvero l'energia di equilibrio della molecola d'acqua.

Da notare come la *chemical accuracy* sperimentale sia di circa $0.04eV/atom$ e variazioni piccole di theta influiscono in minima parte all'energia potenziale del sistema, quindi se si nota la scala delle energie nel grafico A.1 si ha una chiara idea di come ci si sta avvicinando all'ordine di grandezza sopra citato. Si può dire come quindi la rete neurale abbia un errore $\sigma = \frac{0.2eV}{3atoms} = 0.07eV/atom$ simile alla *chemical accuracy* sperimentale in questo problema specifico [15]. Inoltre l'errore è inferiore alla *chemical accuracy* tipica delle simulazioni *DFT*, che rimane costante a circa $\sigma_{DFT} = 0.2eV/atom$, al contrario della *loss function* per le reti neurali [2].

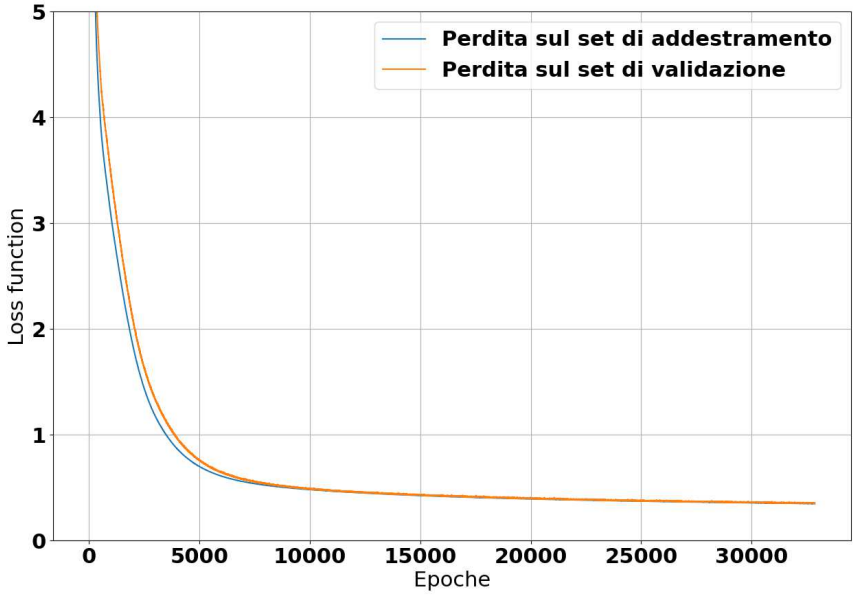


FIGURA 2.2: Errore quadratico medio della rete neurale

2. Osservando invece il grafico relativo alla formazione del legame covalente per H_2 e il grafico relativo alla formazione del gruppo idrossilico OH si nota come l'andamento delle predizioni rispetto alle curve di controllo sia in linea con quanto ci si aspetta dal comportamento della rete con i grafici a parametri liberi.

Il grafico A.4 propone la curva relativa al potenziale del sistema tenendo fisso $\theta = 1.82rad$ e $d_{H2} = 3$Å, in questo modo la variazione del potenziale dipende solo da d_{H1}. In questo caso si nota che come per il grafico A.2 le predizioni seguono bene la curva di controllo e il minimo viene riprodotto con la stessa accuratezza: si ha un minimo di controllo e sperimentale [9] $dOH_{DFT} = dOH_{exp} = 0.97$Å e un minimo trovato tramite predizione di $dOH_{DNN} = 0.97$Å ed energia associata $E_{OH} = -474eV$.

3. Nel caso del grafico per la formazione dell'idrogeno molecolare invece è stata utilizzata una configurazione in cui entrambi gli atomi di idrogeno variano la loro distanza in relazione al valore dell'angolo θ, in questo scenario le predizioni sono quindi più complesse per la rete dato che siamo in uno scenario simile a quello per parametro libero θ.

   La curva viene riprodotta bene in quanto osservando la scala dell'energia vi è un ordine di grandezza in più: qui si vede chiaramente che il minimo per θ corrisponde alla configurazione di legame covalente per H_2. Nonostante ciò il minimo energetico risulta sufficientemente accurato sia con il controllo DFT che con le misure sperimentali [9]:

   $$\theta_{DFT} = 0.37rad \rightarrow dHH_{DFT} = dHH_{exp} = 0.74\text{Å} \qquad \theta_{DNN} = 0.37rad \rightarrow dHH_{DNN} = 0.74\text{Å}$$

   Per calcolare le lunghezze di legame dai valori di θ è stata utilizzata la seguente formula derivante dalla configurazione geometrica degli atomi:

   $$d_{HH} = 2d_{H1} \sin \frac{\theta}{2} \tag{2.3}$$

   Le predizioni delle lunghezze di legame per i minimi energetici rientrano quindi entro il margine d'errore dei rispettivi valori di controllo e sperimentali, con energia $E_{HH} = -473.1eV$.

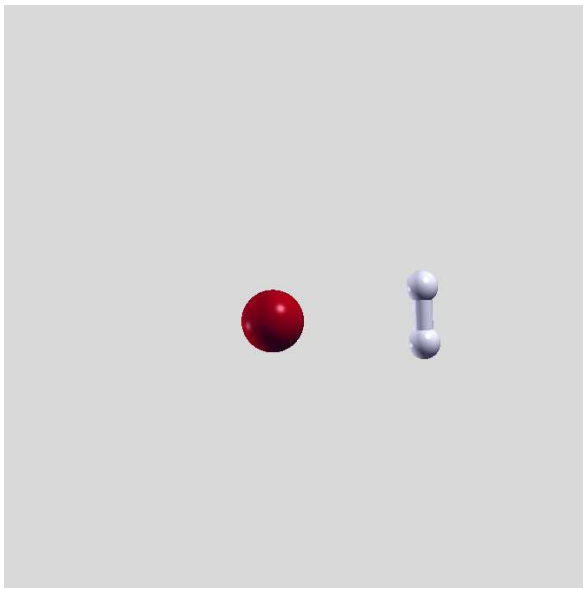


Figura 2.3: Configurazione legame H-H [10]

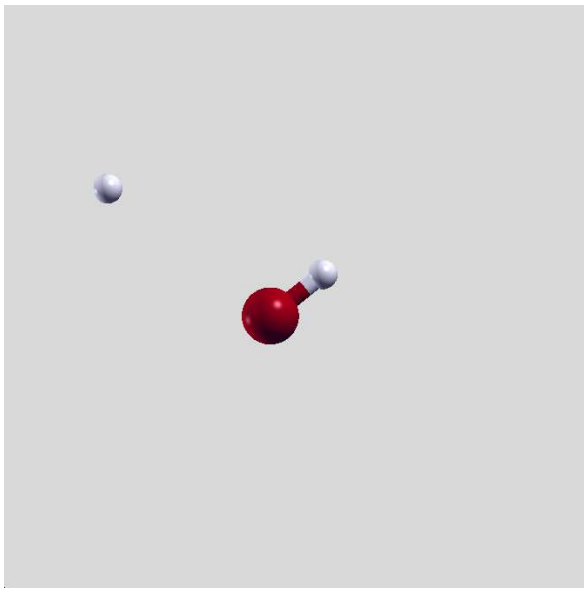


Figura 2.4: Configurazione legame O-H [10]

4. Come ulteriore situazione si può osservare l'ultima configurazione studiata, ovvero la forma del potenziale superficiale per la formazione di entrambi i gruppi molecolari raffigurata nel grafico A.5. Si può vedere come il potenziale in questione abbia due punti di equilibrio: un equilibrio

stabile corrispondente circa a $d_1 = 1$Å e un equilibrio instabile attrattivo corrispondente a circa $d_2 = 2$Å.

Guardando le figure 2.3 e 2.4 si può immaginare una situazione in cui avviene la transizione tra la situazione di destra e quella di sinistra e viceversa tramite lo spostamento di un atomo di idrogeno.

Questi due equilibri corrispondono alla formazione del legame O-H per d_1 e del legame H-H per d_2. Si vede come la curva delle predizioni venga riprodotta con accuratezza e buona precisione anche in questo caso ed entrambi i punti di equilibrio vengono riprodotti bene. Da notare che mentre la d_1 è direttamente la lunghezza di legame in configurazione O-H, la lunghezza d_2 va invece convertita nella distanza di legame H-H tramite la formula di conversione 2.3 utilizzata nel caso del grafico A.3 per confermare l'uguaglianza con il θ fissato.

In figura A.6 si possono notare gli equilibri più nel dettaglio, si vede come le predizioni seguano si la curva di controllo ma essendo la scala dell'energia quattro volte più piccola si vedono maggiormente le discrepanze tra le due curve. Dai due minimi citati sopra si ottengono di nuovo quindi i valori di lunghezza di legami ed energetici ottenuti studiando le due configurazioni separatamente.

Capitolo 3

Conclusioni

Come visto sopra la rete neurale così addestrata si comporta bene in tutte le situazioni presentate, ed è ragionevole pensare che si comporti similmente anche in altre possibili configurazioni diverse da quelle descritte. Da queste si possono trarre alcune conclusioni.

Tramite questi casi fisici è stata analizzata la riproduzione di un parametro derivante dal conto *SCF* tramite rete neurale della casistica specifica dei due atomi di idrogeno e l'atomo di ossigeno. Si può dire come nel caso specifico di questo problema un maggior numero di campioni da inserire nel *dataset* di addestramento possa sicuramente aiutare nel migliorare la precisione e permetta un addestramento di una rete più complessa; rete che a sua volta permetterebbe di coprire casi di configurazioni ancora più variegate e diverse dalle configurazioni casuali immesse.

Per eventuale uso si rimanda alla *repository github* in appendice B con tutto il codice utilizzato e i dati generati. In particolare il modello qui descritto è denominato come: `Adadelta6_6layers_256neurons_35000epochs_128batchsize_0.001l2par.keras`.

Una conclusione interessante, come già detto sopra, è che l'errore ottenuto dalla rete neurale è comparabile alla *chemical accuracy* sperimentale [15], e in questo problema specifico minore della *chemical accuracy* tipica delle simulazioni effettuate tramite *Density Functional Theory* [2].

Si stima che per ottenere una rete neurale funzionale allo scopo di sostituire il calcolo effettuato da `pw.x` per questa specifica configurazione sia necessario un *dataset* di addestramento contenente almeno un ordine di grandezza in più di campioni totali.

Si pensa inoltre che sia possibile addestrare una rete che preveda non solo questo, ma anche gli altri parametri che vengono calcolati tramite il codice `pw.x` come, ad esempio: il numero di orbitali atomici occupati con le relative energie, o anche i singoli contributi energetici all'energia totale del sistema.

Una rete neurale capace di prevedere accuratamente e precisamente le proprietà delle simulazioni *DFT* potrebbe essere quindi utile nei casi in cui la computazione diretta sia svantaggiosa, come ad esempio per molecole molto grandi.

Appendice A

Grafici

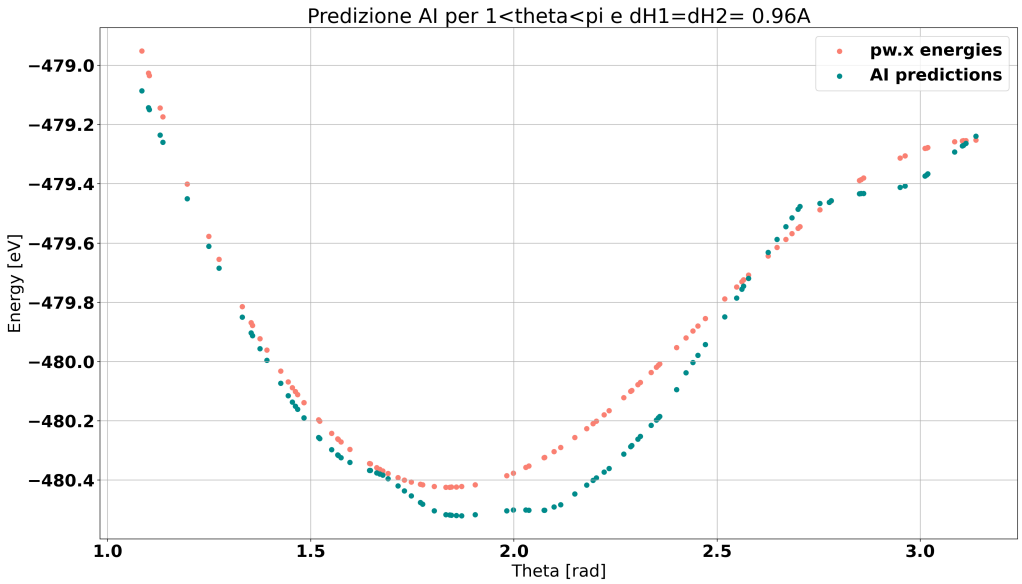


FIGURA A.1: predizione per parametro libero θ

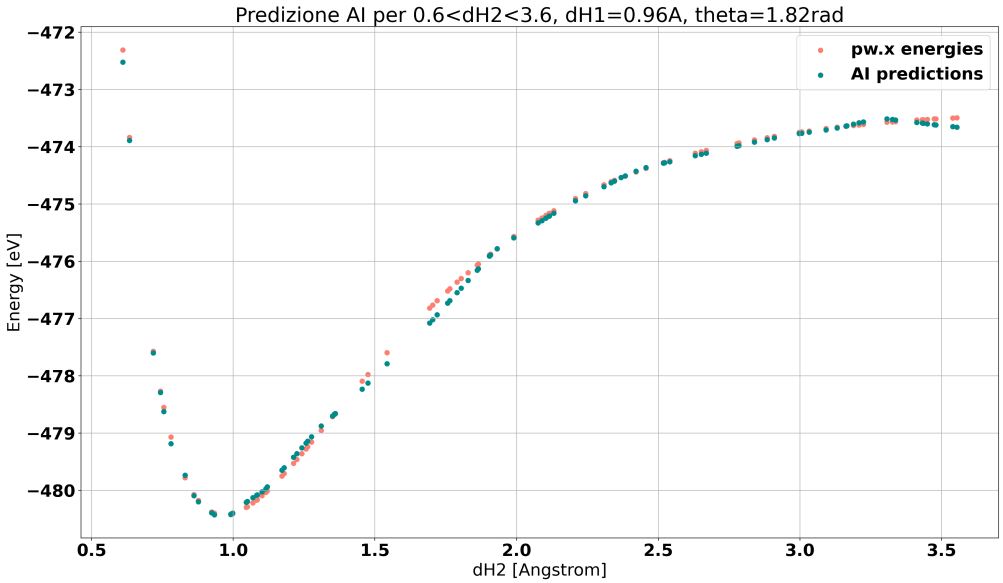


FIGURA A.2: predizione per parametro libero d_{H2}

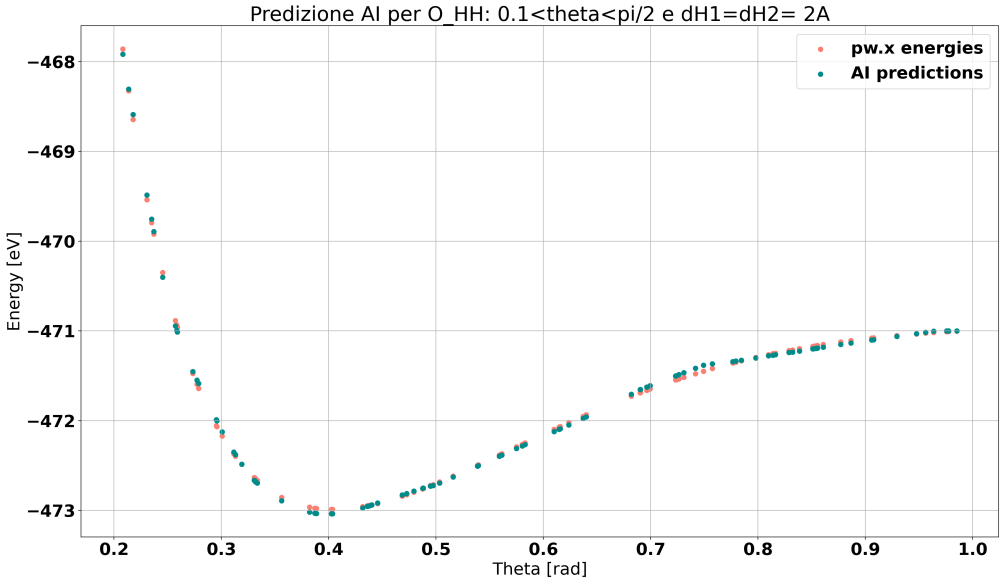


FIGURA A.3: predizione per formazione legame H-H

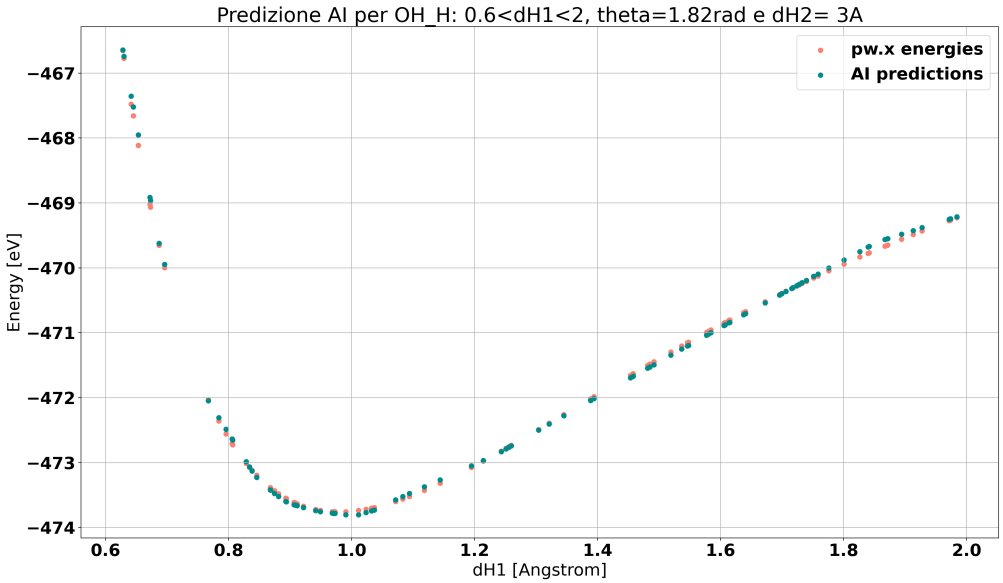


FIGURA A.4: predizione per formazione legame O-H

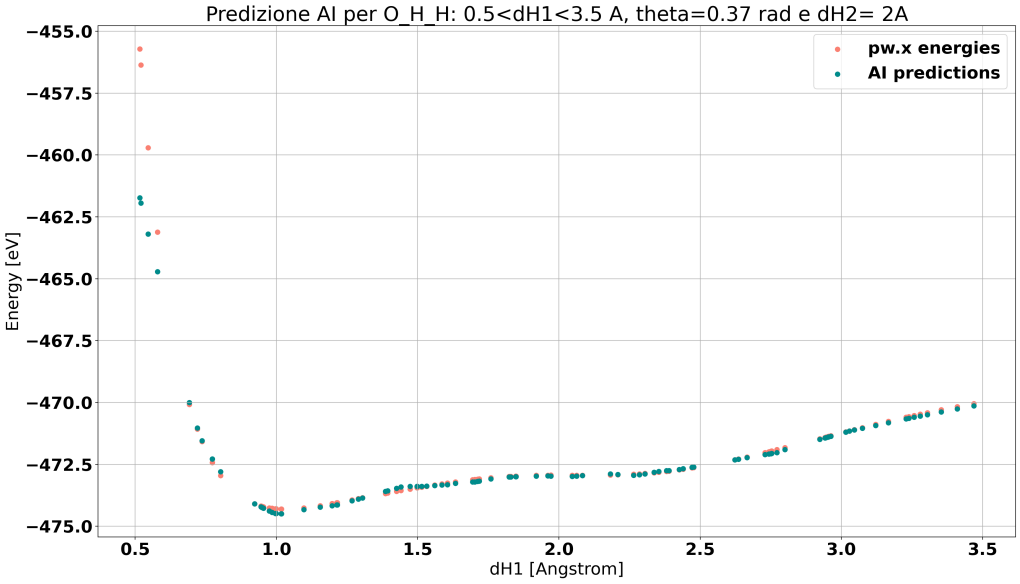


FIGURA A.5: predizione per formazione legami O-H e H-H

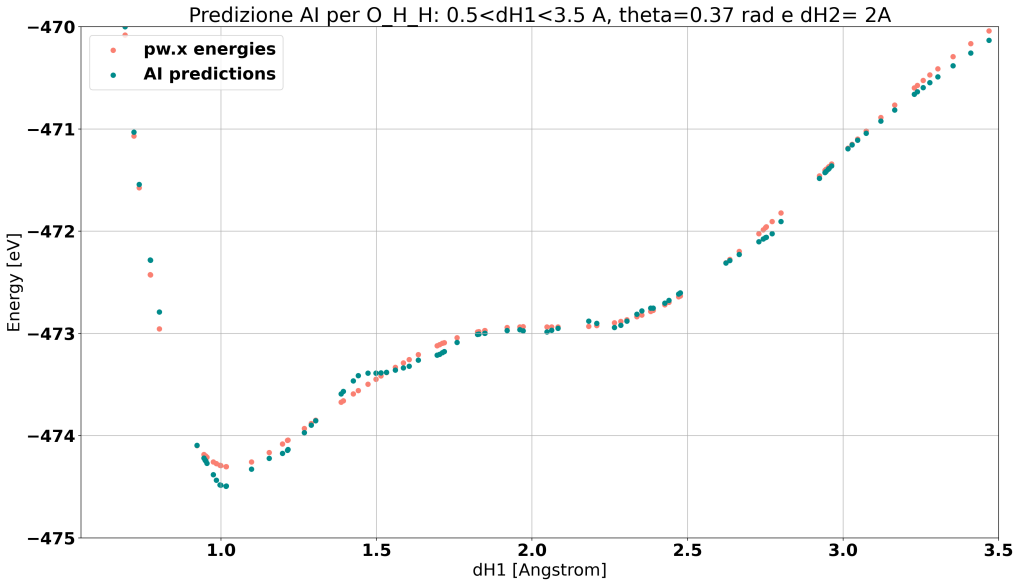


FIGURA A.6: predizione per formazione legami O-H e H-H ingrandita sugli equilibri

Appendice B

Codice utilizzato

Di seguito il collegamento alla *repository* github contenente il codice e il materiale generato durante la ricerca: https://github.com/tusca99/Bachelor-Thesis.git

Bibliografia

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[2] Tuckerman M.E. et al. Bogojeski M. Vogt-Maranto L. *Quantum chemical accuracy from density functional approximations via machine learning. Nat Commun 11, 5223.* 2020. URL: https://doi.org/10.1038/s41467-020-19093-1.

[3] Paolo Giannozzi et al. «Advanced capabilities for materials modelling with Quantum ESPRESSO». In: *Journal of Physics: Condensed Matter* 29.46 (ott. 2017), p. 465901. ISSN: 1361-648X. DOI: 10.1088/1361-648x/aa8f79. URL: http://dx.doi.org/10.1088/1361-648X/aa8f79.

[4] Paolo Giannozzi et al. «QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials». In: *Journal of Physics: Condensed Matter* 21.39 (set. 2009), p. 395502. ISSN: 1361-648X. DOI: 10.1088/0953-8984/21/39/395502. URL: http://dx.doi.org/10.1088/0953-8984/21/39/395502.

[5] Xavier Glorot, Antoine Bordes e Yoshua Bengio. «Deep Sparse Rectifier Neural Networks». In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. A cura di Geoffrey Gordon, David Dunson e Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[6] Ian Goodfellow, Yoshua Bengio e Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[7] Dan Hendrycks e Kevin Gimpel. *Gaussian Error Linear Units (GELUs)*. 2023. arXiv: 1606.08415 [cs.LG].

[8] Sebastiaan P. Huber et al. «AiiDA 1.0, a scalable computational infrastructure for automated reproducible workflows and data provenance». In: *Scientific Data* 7.1 (2020), p. 300. ISSN: 2052-4463. DOI: 10.1038/s41597-020-00638-4. URL: https://doi.org/10.1038/s41597-020-00638-4.

[9] Russell D. Johnson III, cur. *NIST Computational Chemistry Comparison and Benchmark Database, NIST Standard Reference Database Number 101 Release 22*. May 2022. URL: http://cccbdb.nist.gov/.

[10] Anton Kokalj. «XCrySDen—a new program for displaying crystalline structures and electron densities». In: *Journal of Molecular Graphics and Modelling* 17.3 (1999), pp. 176–179. ISSN: 1093-3263. DOI: https://doi.org/10.1016/S1093-3263(99)00028-5. URL: http://www.xcrysden.org/.

[11] lutzroeder. *Netron: Visualizer for neural network, deep learning and machine learning models.* 2024. URL: https://www.lutzroeder.com/ai.

[12] Giovanni Pizzi et al. «AiiDA: automated interactive infrastructure and database for computational science». In: *Computational Materials Science* 111 (2016), pp. 218–230. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2015.09.013. URL: https://www.sciencedirect.com/science/article/pii/S0927025615005820.

[13] Patrick Sit e Linghai Zhang. «Density Functional Theory in Heterogeneous Catalysis». In: *Heterogeneous Catalysts*. John Wiley & Sons, Ltd, 2021. Cap. 23, pp. 405–418. ISBN: 9783527813599. DOI: https://doi.org/10.1002/9783527813599.ch23. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9783527813599.ch23. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9783527813599.ch23.

[14] Martin Uhrin et al. «Workflows in AiiDA: Engineering a high-throughput, event-based engine for robust and modular computational workflows». In: *Computational Materials Science* 187 (2021), p. 110086. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2020.110086. URL: https://www.sciencedirect.com/science/article/pii/S0927025620305772.

[15] Yubo Zhang et al. «Efficient first-principles prediction of solid stability: Towards chemical accuracy». In: *npj Computational Materials* 4.1 (2018), p. 9. ISSN: 2057-3960. DOI: 10.1038/s41524-018-0065-z. URL: https://doi.org/10.1038/s41524-018-0065-z.