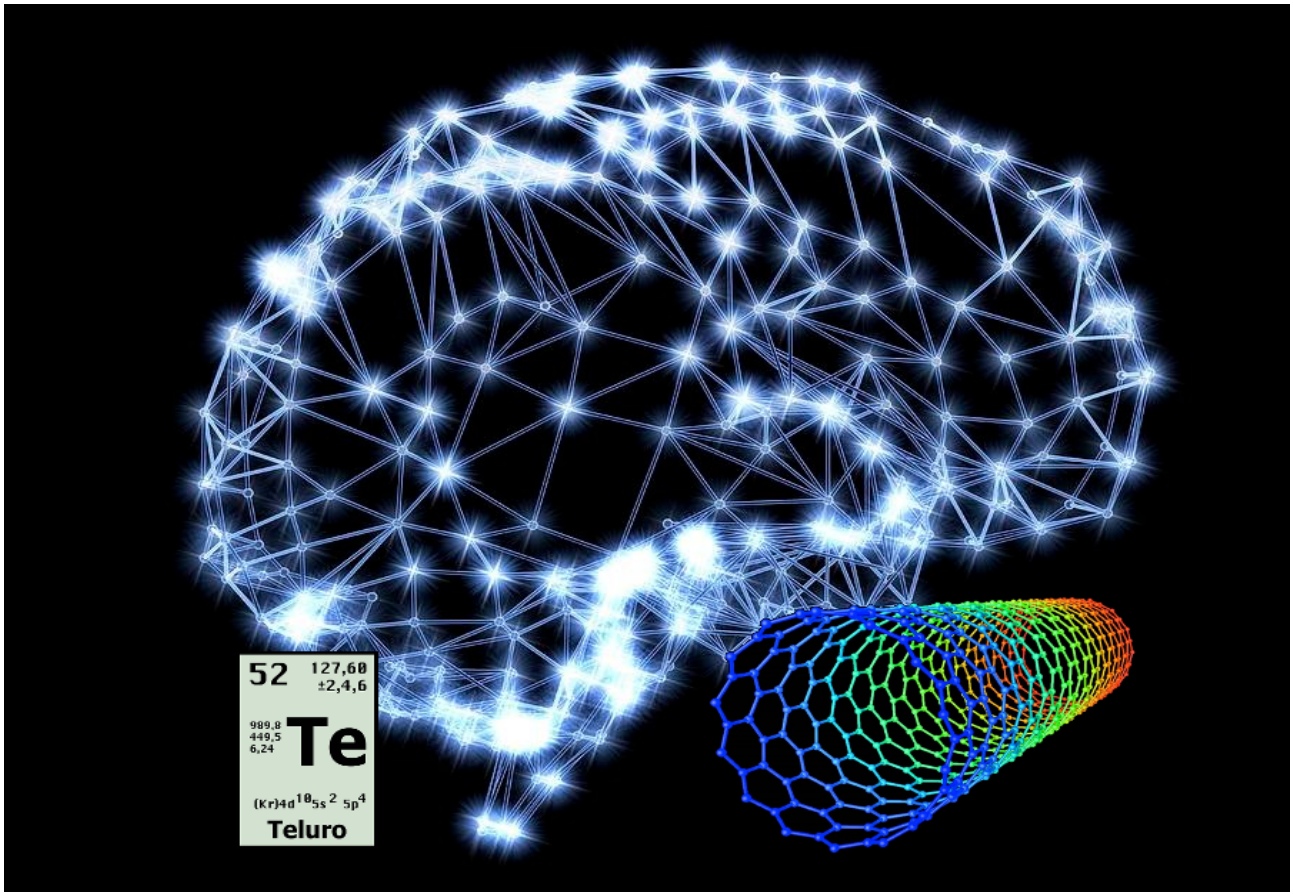
High-Dimensional Neural Network Potentials



Antonio Díaz Pozuelo

adpozuelo@uoc.edu

TFG – Grado en Ingeniería Informática

Universitat Oberta de Catalunya

Enero de 2016

Antonio Díaz Pozuelo – adpozuelo@uoc.edu

ÍNDICE

Antonio Díaz Pozuelo – adpozuelo@uoc.edu

INTRODUCCIÓN

El objetivo del presente TFG (Trabajo Final de Grado) consiste en diseñar e implementar una red neuronal pre-alimentada FFNN (*Feed-Forward Neural Network*) sobre una GPU (*Graphical Processor Unit*) del fabricante NVIDIA con arquitectura CUDA (*Compute Unified Device Architecture*). Dicha red neuronal debe ser capaz de aprender a partir de las propiedades de determinados conjuntos de sistemas de átomos (posiciones en una secuencia temporal y energía para una temperatura y densidad dadas) y predecir las propiedades macroscópicas de otro sistema de átomos diferente, en condiciones de temperatura y/o densidad, a los utilizados para el aprendizaje.

Mediante dicha red neuronal se conseguirá:

- Explotar la tecnología GPGPU (*General Purpose Graphical Processor Unit*) hacia la que se dirige el cálculo científico inexorablemente.
- Emplear la técnica FFNN *(Feed-Forward Neural Network)* de la inteligencia artificial para diseñar un sistema inteligente capaz de aprender y predecir propiedades físicas.
- Reducir el tiempo de cálculo requerido hasta ahora, mediante las técnicas basadas en mecánica cuántica, para obtener las propiedades macroscópicas del sistema a predecir.

A continuación se resume el contenido de esta memoria:

- En el primer capítulo se detallará la planificación del proyecto con el objeto de realizar una gestión del mismo de la forma más eficiente posible.
- En el segundo capítulo se expondrá el problema a afrontar y su análisis de forma detallada. A continuación se detallará la solución propuesta para un entendimiento del dominio del problema. Además se expondrá el motivo por el cual se ha decidido utilizar la arquitectura CUDA.
- En el tercer capítulo se realizará el diseño de la aplicación que aplica la solución elegida en el capítulo anterior. El artefacto de esta fase será un algoritmo con topología de arriba a abajo (*top-down*) en el que se detallará tanto las fases que se ejecutarán en paralelo como sus parámetros (bloques e hilos) y relaciones (memoria compartida o global).
- En el cuarto capítulo se detallará la implementación del artefacto procedente del diseño, explicando las decisiones de implementación tomadas. El artefacto de esta fase será el código fuente de la aplicación en su versión final.
- En el quinto capítulo se presentarán los resultados de las pruebas y de las simulaciones realizadas. Referente a las simulaciones se presentarán resultados GPU vs GPU (utilizando diferentes familias de GPU), utilizando diferentes tamaños de entrada (más o menos conjuntos de entrenamiento), etc.
- En el sexto capítulo se expondrán las conclusiones realizando un análisis de los datos obtenidos en el capítulo anterior.
- Finalmente, los últimos capítulos harán referencia a los anexos, a los agradecimientos y a la bibliografía.

CAPÍTULO 1

PLANIFICACIÓN

Para realizar este proyecto se utilizará un enfoque metodológico en cascada, ya que cada etapa del proyecto nos dará como resultado un artefacto que se utilizará en la siguiente fase. Por lo tanto, los artefactos de cada etapa nos marcarán un desarrollo lineal e iterativo, con objeto de corregir y ajustar las fases según se desarrolle el proyecto.

Podemos dividir el desarrollo del proyecto en varias etapas:

- Análisis y propuesta de soluciones: se expone el problema a afrontar y la solución adoptada dentro de las posibles existentes. El artefacto final será una guía que se basará en la selección de los artículos mediante los cuales se realizará el diseño final del producto.
- Diseño: tras definir como abordar el problema se diseñará un algoritmo "*top-down*" que resuelva el problema. Los artefactos intermedios serán los algoritmos de cada módulo dentro del programa y el artefacto final consistirá en la formulación completa del algoritmo.
- Implementación: en esta fase se implementará el algoritmo definido en la fase anterior. Mediante los algoritmos intermedios se codificarán los módulos y se corregirá el diseño según se precise. Una vez todos los módulos funcionen correctamente se ensamblarán con objeto de dar forma al artefacto definitivo, la versión final del programa en código fuente.
- Pruebas: es la etapa en la que se ejecutan los diversos módulos implementados para comprobar su correcto funcionamiento y rendimiento. Finalmente, se ejecutará el artefacto final de la fase de implementación en diferentes entornos (GPU vs GPU) y con diferentes parámetros (tamaños de los conjuntos de entrada). El artefacto final corresponderá a una serie de métricas que representarán las diversas ejecuciones realizadas.
- Mantenimiento: en esta fase se utilizarán las métricas anteriores para proponer mejoras y correcciones a los posibles defectos de la aplicación.

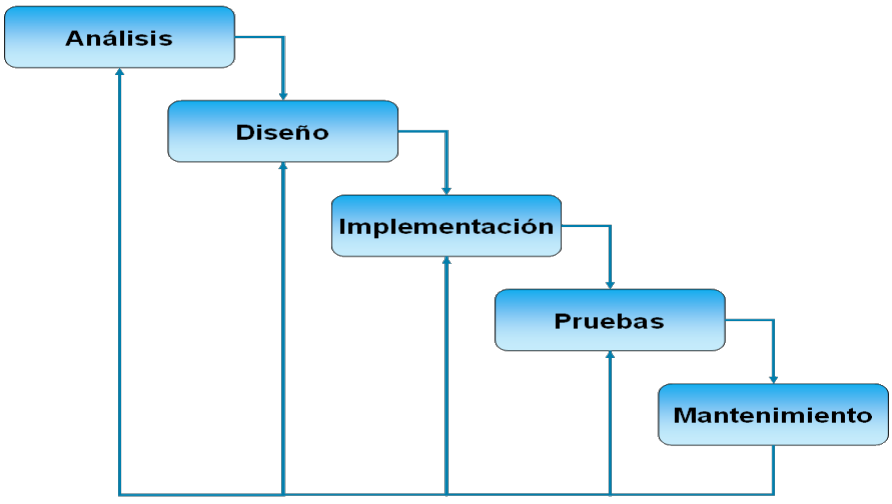


Figura 1: método cascada.

1.1 - Diagrama de Gantt:

A continuación, se expone el diagrama de Gantt asociado al proyecto:

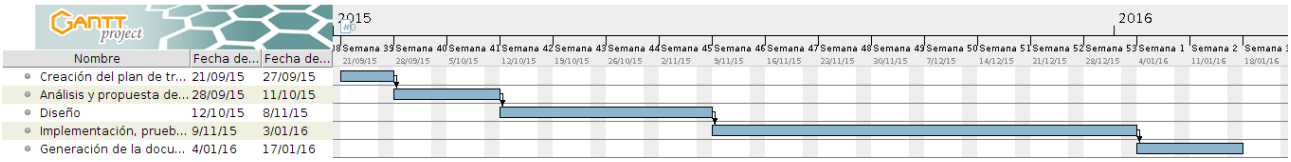


Figura 2: diagrama de Gantt HDNNP.

1.2 – Planificación detallada:

ID	Tarea	FIN	Duración	Comienzo	Final
1	Creación del plan de trabajo	X	7 días	21/09/2015	27/09/2015
2	Obtener la documentación necesaria para el proyecto	X	3 días	23/09/2015	26/09/2015
3	Instalación y puesta en marcha del entorno de trabajo	X	7 días	21/09/2015	27/09/2015
4	Análisis y propuesta de soluciones	X	14 días	28/09/2015	11/10/2015
5	Estudio y análisis del problema	X	7 días	28/09/2015	03/10/2015
6	Estudio y análisis de las soluciones	X	7 días	04/10/2015	11/10/2015
7	Diseño → Implementación → Pruebas [Método cascada	X	76 días	12/10/2015	27/12/2015
8	Algoritmo general	X	3 días	12/10/2015	15/10/2015
9	Módulo 1 → Control argumentos de entrada	X	4 días	16/10/2015	20/10/2015
10	Módulo 2 → Proceso de ficheros de entrada	X	6 días	21/10/2015	27/10/2015
11	Módulo 3 → Pre-condicionado de datos	X	21 días	28/10/2015	18/11/2015
12	Módulo 4 → Red neuronal (aprendizaje)	X	33 días	19/11/2015	21/12/2015
13	Módulo 5 → Red neuronal (predicción)	X	3 días	22/12/2015	25/12/2015
14	Módulo 6 → Proceso de ficheros de salida	X	1 día	26/12/2015	27/12/2015
15	Simulaciones	X	7 días	28/12/2015	03/01/2016
16	Análisis de las simulaciones	X	3 días	04/01/2016	07/01/2016
17	Documentación	X	13 días	08/01/2016	21/01/2016

CAPÍTULO 2

ANÁLISIS Y PROPUESTA DE SOLUCIONES

2.1 - Exposición del problema:

Dado un sistema de átomos (configuración) se quieren calcular las propiedades macroscópicas (energía, presión, conductividad, etc.) del mismo.

2.2 - Posibles soluciones:

1. Por un lado existen soluciones a este problema utilizando interacciones efectivas entre pares/tripletes de partículas ajustadas a cálculos mecano-cuánticos y/o propiedades experimentales. Éste es un procedimiento habitual cuando no se requiere gran precisión en sistemas relativamente simples. Esta solución no permite el estudio de reacciones químicas ni de elementos semi-conductores (p.e. el Teluro).

2. Por otro lado, se puede resolver la ecuación de Schrödinger para un conjunto de átomos dado - empleando la teoría del funcional de la densidad (DFT) - y obtener las energías y las fuerzas correspondientes. Estos métodos, conocidos como *ab initio*, son computacionalmente muy costosos y sólo pueden aplicarse a unos cientos de átomos.

3. Finalmente, se puede solucionar este problema utilizando redes neuronales NN (*Neural Networks*) dada su habilidad para representar funciones arbitrarias. Por un lado, estas redes se consideran "aproximadores universales", de modo que permiten aproximar funciones desconocidas multidimensionales a una precisión arbitraria basada en un conjunto de valores de funciones conocidos. Por otro lado, estas redes aprenden a partir de datos *ab initio* para conjuntos pequeños de átomos permitiendo un estudio detallado en condiciones donde el cálculo requiere muestras (número de átomos/moléculas) grandes.

   Concretamente, y para el problema dado, las NNs tienen las siguientes ventajas:

   1. Las energías se pueden ajustar con gran precisión a cálculos *ab initio* o experimentales.
   2. Requieren mucho menos tiempo de CPU que los cálculos *ab initio*.
   3. No se requiere conocimiento previo sobre la forma funcional de la superficie de potencial/energía.
   4. La expresión de la energía es imparcial, generalmente aplicable a todos los tipos de sistemas y no requiere de modificaciones específicas del sistema.

Sin embargo, las NNs también poseen las siguientes desventajas:

1. La evaluación de las NNs es notablemente más lenta que el uso de campos de fuerza clásicos simples.

2. Las NNs no tienen base física y muy limitadas capacidades de extrapolación.
3. La construcción de NNs requiere de un esfuerzo sustancial.
4. Actualmente, las NNs están limitadas a sistemas que contienen o muy pocos elementos químicos distintos con muchos átomos o a pocos átomos y un número de especies químicas arbitrarias.

2.3 - Solución elegida:

Para resolver el problema dado se utilizará la solución ofrecida por las redes neuronales debido a que:

1. Se desea aprender sobre el diseño y desarrollo de redes neuronales.
2. Es el campo menos desarrollado de los tres citados anteriormente.
3. Existe la necesidad en el Instituto de Química Física Rocasolano, del Consejo Superior de Investigaciones Científicas, de disponer de esta metodología. Por lo tanto, esta solución podrá tener un carácter práctico en el futuro.

Además, se tendrán las siguientes consideraciones:

1. La red neuronal será pre-alimentada FFNN (*Feed-Forward Neural Network*) de modo que sólo se alimentará de los datos de aprendizaje una vez por ejecución y las conexiones entre neuronas no formarán ciclos.
2. La red neuronal será multi-capa (contendrá varias capas ocultas) con el objeto de obtener una mayor precisión, flexibilidad y capacidad de ajuste.
3. La red neuronal se construirá para sistemas de altas dimensiones, esto es: sistemas que contienen cientos o miles de átomos con todos sus grados de libertad explícitos.

Finalmente, se utilizarán como guías de diseño los siguientes artículos:

1. “Constructing High-Dimensional Neural Network Potentials: A Tutorial Review”, Jörg Behler. *Int. J. Quantum Chem*. **2015**, *115*, 1032-1050. DOI: 10.1002/qua.24890
2. “Neural Network Models of Potential Energy Surfaces: Prototypical Examples”, James B. Witkoskie and Douglas J. Doren. *Journal of Chemical Theory and Computation.* **2005**, *1 (1)*, 14-23. DOI: 10.1021/ct049976i
3. Trabajos relacionados:
    1. “Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces”, Behler y Parrinello, Phys. Rev. Lett. **2007**, *98*, 146401; DOI: 10.1103/PhysRevLett.98.146401
    2. “Neural network models of potential energy surfaces”, Thomas B. Blank, Steven D. Brown, August W. Calhoun, and Douglas J. Doren. J. Chem. Phys. **1995,** *103*, 4129; DOI: 10.1063/1.469597.
    3. “Atom-centered symmetry functions for constructing high-dimensional neural network potentials”, Jörg Behler. J. Chem. Phys. **2011**, *134*, 074106; DOI: 10.1063/1.3553717.

2.4 - Arquitectura computacional y tecnología:

La construcción de la solución elegida se puede realizar de forma secuencial en su totalidad o paralelizando las partes que lo permitan. Evidentemente, se optará por paralelizar las partes que admitan este paradigma con objeto de obtener el mejor rendimiento posible.
Respecto a la paralelización, se puede optar por paralelizar utilizando la CPU o la GPU. La paralelización mediante CPU se puede considerar una tecnología "clásica" mientras que la paralelización mediante GPU es una tecnología "actual" y en auge.

Por lo tanto, se optará por la paralelización (de las partes de la solución que lo admitan) utilizando la GPU dado que:

1. Se desea aprender el desarrollo de aplicaciones utilizando esta tecnología.
2. La paralelización en GPU es una tecnología en continua expansión en el ámbito científico.
3. La eficiencia de las GPUs es superior respecto de las CPUs.

Además, se decide utilizar GPUs de la marca NVIDIA con arquitectura CUDA dado que:

1. Se desea aprender a implementar aplicaciones en CUDA_C/C++.
2. Se trata de GPUs de ámbito general que se pueden encontrar en gran parte del ecosistema informático existente.
3. Disponen de un SDK (*Software Development Kit*) maduro y basado en el lenguaje de programación C.

Finalmente, se utilizarán como guías de estudio CUDA_C/C++ los siguientes recursos:

1. CUDA Programming - ISBN: 978-0-12-415933-4.
2. CUDA By Example - ISBN 978-0-13-138768-3.
3. CUDA Toolkit Documentation – http://docs.nvidia.com/cuda/index.html.

CAPÍTULO 3

DISEÑO

El diseño de la aplicación se va a realizar de forma modular por lo que cada parte de la solución se pueda diseñar e implementar en base a descomposiciones (especificaciones) de otras partes más genéricas.

Dada la alta cantidad de información a procesar y almacenar, cada módulo deberá ser lo máximo posible independiente de los otros con objeto de optimizar los recursos (memoria RAM). Esto permitirá realizar ejecuciones independientes de cada módulo obteniendo resultados de las pruebas que retro-alimentarán el ciclo diseño → implementación → pruebas.

3.1 – Algoritmo general:

En base a lo expuesto anteriormente, se procede a describir la primera versión del algoritmo y a descomponerlo mediante un procedimiento "*top-down*":

Algoritmo_HDNNP v:0.1 → High-Dimensional Neural Networks Potentials Application
Datos de entrada → temperaturas de los sistemas para aprender y a predecir.
Datos de salida → energías de los sistemas a predecir.

comienzo_algoritmo
[Módulos independientes {1, 2, 3}]
1. Control de argumentos de entrada.
2. Proceso de ficheros de entrada.
3. Pre-condicionado de datos para la red neuronal.

[Módulos independientes {4, 5, 6}]
4. Ejecución de la red neuronal (modo aprendizaje).
5. Ejecución de la red neuronal (modo predicción).
6. Proceso de ficheros de salida.

fin_algoritmo

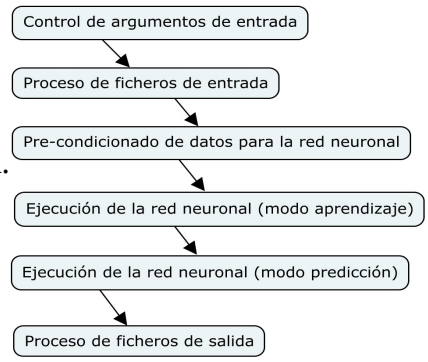


Figura 3: algoritmo HDNNP v:0.1

El grupo de módulos 1, 2 y 3 se pueden ejecutar independientemente del grupo de módulos 4, 5 y 6. Éstos últimos también se pueden ejecutar de forma independiente de los primeros siempre y cuando éstos se hayan ejecutado al menos una vez. Por lo tanto, se puede procesar el grupo de módulos 1, 2 y 3 una única vez (ya que siempre se realizan las mismas operaciones) y posteriormente ejecutar la red neuronal con diferentes configuraciones tantas veces como sea necesario.

Los datos de entrada y de salida se detallan en el anexo de la memoria.

3.2 - Módulo 1 → Control de argumentos de entrada:

El objetivo de este módulo es controlar los argumentos de entrada de la aplicación y, mediante dichos argumentos, generar los nombres de los ficheros de entrada a procesar. Por lo tanto, se deben generar, por cada temperatura de entrada dos nombres de ficheros: uno para el fichero (*history*) que contiene la secuencia temporal de conjuntos de átomos y otro (*energy*) que contiene la energía, ya calculada mediante CPU utilizando simulaciones mecano-cuánticas, de cada sistema de átomos.

Concretamente, para los argumentos de sistemas de átomos a predecir los nombres de ficheros creados relativos a la energía se corresponderán con el nombre de los ficheros de salida del programa. Esto es: serán los ficheros en donde el programa escribirá los resultados de las predicciones.

Por lo tanto, para el control de argumentos de entrada, se aplicarán las especificaciones y limitaciones descritas en el apartado de los datos de entrada (Anexo A.1).

3.2.1 – Algoritmo:

A continuación se detalla el algoritmo del módulo:

Algoritmo_HDNNP v:0.2; Módulo 1 → Control de argumentos de entrada
    Datos de entrada → argumentos de entrada.
    Datos de salida → nombres de los ficheros de entrada a procesar.

comienzo_algoritmo
    Si hay menos de dos argumentos, salir del programa
    Mientras existan argumentos:
        Si la expresión regular (“^(l|p)[0-9]{1,4}$”) se cumple:
            Contar argumentos a aprender y a predecir
        En caso contrario, salir del programa
    Si los argumentos a aprender o a predecir son iguales a cero, salir del programa
    Mientras existan argumentos:
        Si el argumento contiene una “l”
            Crear y almacenar el nombre de fichero “data/HISTORY.Te.[?].K” ([?] → dato de entrada sin la “l”)
            Crear y almacenar el nombre de fichero “data/E.Te.[?].K” ([?] → dato de entrada sin la “l”)
        Si el argumento contiene una “p”
            Crear y almacenar el nombre de fichero “data/HISTORY.Te.[?].K” ([?] → dato de entrada sin la “p”)
            Crear y almacenar el nombre de fichero “data/E.Te.[?].K” ([?] → dato de entrada sin la “p”)
fin_algoritmo

La ejecución de este algoritmo será realizada de forma secuencial por la CPU.

3.3 - Módulo 2 → Proceso de ficheros de entrada:

Antonio Díaz Pozuelo – adpozuelo@uoc.edu

El objetivo de este módulo es leer los distintos ficheros de entrada de aprendizaje y de predicción, que contienen todas las secuencias temporales de los conjuntos de átomos y sus energías, y crear un único fichero de salida compuesto por la energía de cada secuencia temporal (conjunto de átomos) junto a los datos estrictamente necesarios de dicha secuencia.

Por un lado, el fichero que contiene la secuencia temporal ("data/HISTORY.Te.[?].K") está compuesto por conjuntos de átomos. Cada conjunto de átomos se encuentra descrito por su número (*timestep*), por el número de átomos que contiene, por el tamaño (Å) de la caja de simulación (*box*) en la que están los átomos y por la descripción de cada átomo dentro de la caja. A su vez, cada átomo esta descrito por su nombre, un número que le identifica, su peso molecular (u.m.a), sus coordenadas (Å), su velocidad (Å / ps) y su fuerza (u.m.a * Å / ps^2).

```
timestep          1        64         2         1    0.001000
   14.508000      0.000000      0.000000
    0.000000     14.508000      0.000000
    0.000000      0.000000     14.508000
      Te          1  127.600000    0.000000
    9.2039780617E+00      9.7824159389E+00      2.8602546664E+00
    6.9020853329E-01      5.7969950169E-01     -5.0218420389E+00
    2.1141000000E-01      7.3537400000E-01     -2.0765000000E-01
      Te          2  127.600000    0.000000
    4.9346167759E+00      2.1517051280E+00      4.2864128284E+00
   -6.4354145474E-01      3.1454012737E+00     -7.9618288613E-01
    1.6940500000E-01      6.5414700000E-01      2.2044600000E-01
      Te          3  127.600000    0.000000
    1.0175378176E+01      4.5515028593E+00      1.3390518543E+01
    5.0135202338E-01     -5.8780197567E-01     -1.1360254390E+00
   -2.0113400000E-01      5.2877400000E-01      5.7874100000E-01
      Te          4  127.600000    0.000000
    9.8613298836E-01      9.5423831100E+00      1.4415460722E+00
    1.0302747776E+00     -3.7611708550E+00      1.1845292580E+00
    6.0212800000E-01     -1.2605300000E-01     -2.3624700000E-01
      Te          5  127.600000    0.000000
    1.0231880453E+01      7.7367275856E-01      6.2895434144E+00
   -2.0277714305E+00     -1.4420043594E+00     -2.8902172356E+00
    5.0040300000E-01      2.7809400000E-01      1.2903300000E-01
      Te          6  127.600000    0.000000
    1.0756549941E+01      2.8265929146E+00      1.0011055345E+01
   -1.5319025498E+00     -1.4049868508E+00      2.0238728085E+00
   -3.4564200000E-01      1.6534200000E-01      4.2873300000E-01
      Te          7  127.600000    0.000000
    8.9678764656E+00      6.6238054132E+00      4.7118632652E-01
   -3.4923324006E+00      1.7661316123E-01      1.5661413212E-01
    2.8351200000E-01     -2.8377000000E-01     -3.7302300000E-01
```

Fichero HISTORY.Te.673K.

Por otro lado, el fichero que contiene la energía de cada conjunto de átomos ("data/E.Te.[?].K") está compuesto por el tiempo de cada paso que ocupa el conjunto (en ps), dentro de la secuencia temporal, y de la energía (en eV) de dicho conjunto de átomos.

```
0.000000   -185.488060
0.001000   -185.470680
0.002000   -185.453920
0.003000   -185.437850
0.004000   -185.422500
0.005000   -185.407910
0.006000   -185.394180
0.007000   -185.381330
0.008000   -185.369410
0.009000   -185.358520
0.010000   -185.348730
0.011000   -185.340070
0.012000   -185.332590
0.013000   -185.326300
0.014000   -185.321260
0.015000   -185.317470
0.016000   -185.314960
```

Fichero E.Te.673K.

3.3.1 – Selección de características:

Se debe realizar una la selección de características mediante la cual se reducirá la dimensionalidad del conjunto de datos y se escogerá únicamente aquellas características que se necesiten. Concretamente, para el fichero que contiene las secuencias temporales de átomos, sólo se seleccionarán las siguientes características:

1. Número de átomos que contiene la caja: es un dato necesario dado que se utilizará para inicializar estructuras de datos, controlar bucles, etc. Además, se debe asegurar que todas las cajas contienen el mismo número de átomos.
2. Dimensiones de la caja: las condiciones de contorno incluyen periodicidad, por lo que cada caja de átomos es periódica en el espacio.
3. Coordenadas del átomo: describen la posición de cada átomo en el espacio (x, y, z) respecto a un vértice de la caja de simulación.
4. Fuerza ejercida sobre el átomo: mediante la fuerza se sabrá si los valores de representación obtenidos de evaluar las funciones de simetría (se verá este concepto más adelante) representan correctamente a dicho átomo. Además, este dato será útil para mejoras de rendimiento en el sistema de optimización de parámetros de la red neuronal y para usarlo como dato supervisado en la predicción de la fuerza de otros sistemas.

Como en el caso anterior, y para el fichero de energías de las cajas de átomos, se procederá a realizar una selección de características, seleccionando las siguientes:

1. Energía del conjunto de átomos: se trata de la salida supervisada de la red neuronal cuando se ejecuta en modo de aprendizaje. Mediante este dato se podrá estimar el error cometido en cada época (error de predicción) de la red neuronal.

Para ambos casos, fichero de secuencia temporal y fichero de energía, se debe eliminar la primera fila del mismo ya que contiene datos no necesarios.

Todos los ficheros de entrada han sido cedidos por el departamento de Mecánica Estadística y Materia Condensada del Instituto de Química Física Rocasolano (CSIC). Estos datos de entrada incluyen la secuencia temporal de las posiciones atómicas del sistema de aprendizaje. Dichos datos proceden de simulaciones *ab initio* para Teluro fundido a varias temperaturas, realizadas con el software VASP (*Vienna Ab initio Simulation Package*) y proporcionadas por el Dr. Nebil A. Katcho (CIC Energigune, Álava).

3.3.2 – Ficheros de salida y sesgo temporal:

Finalmente, con las características seleccionadas anteriormente se creará un fichero de salida para el aprendizaje (“data/LEARNING_DATA.dat”) y otro para la predicción (“data/PREDICT_DATA.dat”), los cuales contendrán cada uno de los conjuntos de átomos representados con los datos exclusivamente necesarios: la energía del conjunto, el tamaño de la caja y la posición y fuerza sobre cada átomo.

```
-185.488068
14.5080003738 14.5080003738 14.5080003738
9.2039785385e+00 9.7824163437e+00 2.8602547646e+00
2.1141000092e-01 7.3537397385e-01 -2.0765000582e-01
4.9346165657e+00 2.1517050266e+00 4.2864127159e+00
1.6940499842e-01 6.5414702892e-01 2.2044600546e-01
1.0175377846e+01 4.5515027046e+00 1.3390518188e+01
-2.0113399625e-01 5.2877402306e-01 5.7874101400e-01
9.8613297939e-01 9.5423831940e+00 1.4415460825e+00
6.0212802887e-01 -1.2605300546e-01 -2.3624700308e-01
1.0231880188e+01 7.7367275953e-01 6.2895436287e+00
5.0040298700e-01 2.7809399366e-01 1.2903299928e-01
1.0756549835e+01 2.8265929222e+00 1.0011054993e+01
-3.4564200044e-01 1.6534200311e-01 4.2873299122e-01
8.9678764343e+00 6.6238055229e+00 4.7118633986e-01
2.8351199627e-01 -2.8376999497e-01 -3.7302300334e-01
```

Fichero LEARNING_DATA.dat.

Cabe señalar que la lectura de los conjuntos de átomos de los ficheros de entrada se realiza de forma secuencial por la propia naturaleza de dichos ficheros. Esto conlleva que la escritura de dichos conjuntos tras la selección de características se realizaría en el mismo orden secuencial de la lectura, creando un sesgo temporal relativo a las temperaturas seleccionadas. Con objeto de evitar dicho sesgo en el aprendizaje se debe realizar un intercalado, en la escritura del fichero de salida para el aprendizaje, entre los conjuntos de átomos de las distintas secuencias temporales. De esta forma el fichero de salida contendrá un ciclo compuesto de una configuración de átomos de la primera secuencia temporal (primera temperatura), luego de la segunda, luego de la tercera y así sucesivamente hasta volver a contener una configuración de la primera secuencia temporal (figura 4).

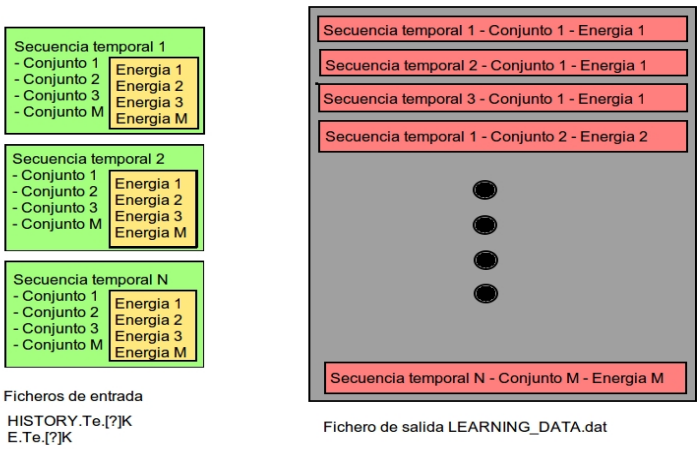


Figura 4: estructura del fichero de salida del aprendizaje.

3.3.3 – Algoritmo:

A continuación se detalla el algoritmo del módulo:

Algoritmo_HDNNP v:0.3; Módulo 2 → Proceso de ficheros entrada
Datos de entrada → nombres de los ficheros de entrada a procesar.
Datos de salida → ficheros “data/LEARNING_DATA.dat” y “data/PREDICT_DATA.dat”

```
comienzo_algoritmo
    [Procesar ficheros de aprendizaje]
    Abrir fichero de salida en modo escritura (“data/LEARNING_DATA.dat”)
    Abrir ficheros de entrada de aprendizaje en modo lectura utilizando un vector de punteros
    Descartar la primera línea de los ficheros de entrada
    Mientras los ficheros de entrada contengan datos
        Recorrer el vector de punteros de ficheros de entrada (intercalado)
            Leer la energía del fichero de entrada de energía (“data/E.Te.[?].K”) y
            escribirla en el fichero de salida “data/LEARNING_DATA.dat”
            Localizar en el fichero de entrada de secuencia temporal (“data/HISTORY.Te.
            [?].K”) una configuración (timestep)
                Leer el número de átomos de la caja del fichero de entrada
                Si el número de átomos no es igual en todas las cajas, salir del
                programa
                Leer las dimensiones de la caja del fichero de entrada y escribirlas en
                el fichero de salida “data/LEARNING_DATA.dat”
                Para todos los átomos en la caja
                    Leer las coordenadas del átomo del fichero de entrada y
                    escribirlas en el fichero de salida “data/LEARNING_DATA.dat”
                    Leer las fuerzas del átomo del fichero de entrada y escribirlas
                    en el fichero de salida “data/LEARNING_DATA.dat”
    Cerrar los ficheros de entrada y salida
    [Procesar ficheros de predicción]
    Abrir fichero de salida en modo escritura (“data/PREDICT_DATA.dat”)
    Abrir ficheros de entrada de predicción en modo lectura utilizando un vector de punteros
    Descartar la primera línea de los ficheros de entrada
    Mientras los ficheros de entrada contengan datos
        Leer la energía del fichero de entrada de energía (“data/E.Te.[?].K”) y
        escribirla en el fichero de salida “data/PREDICT_DATA.dat”
        Localizar en el fichero de entrada de secuencia temporal (“data/HISTORY.Te.
        [?].K”) una configuración (timestep)
            Leer el número de átomos de la caja del fichero de entrada
            Si el número de átomos no es igual en todas las cajas, salir del
            programa
            Leer las dimensiones de la caja del fichero de entrada y escribirlas en el
            fichero de salida “data/PREDICT_DATA.dat”
            Para todos los átomos en la caja
                Leer las coordenadas del átomo del fichero de entrada y
                escribirlas en el fichero de salida “data/PREDICT_DATA.dat”
                Leer las fuerzas del átomo del fichero de entrada y escribirlas
                en el fichero de salida “data/PREDICT_DATA.dat”
    Cerrar los ficheros de entrada y salida
fin_algoritmo
```

La ejecución de este algoritmo será realizada de forma secuencial por la CPU.

3.3.4 - Posibles mejoras:

La lectura de los ficheros de entrada se puede paralelizar mediante el uso del sistema de ficheros PVFS y/o la librería MPI/IO.

3.4 - Módulo 3 → Pre-condicionado de datos para la red neuronal:

El objetivo de este módulo es leer las configuraciones de átomos de los ficheros "data/LEARNING_DATA.dat" y "data/PREDICT_DATA.dat", y realizar las siguientes operaciones:

1. Generar los valores de representación de cada átomo mediante funciones de simetría.
2. Seleccionar las características exclusivamente necesarias.
3. Comprobar que los valores de representación de los átomos caracterizan correctamente la configuración.
4. Normalizar los datos de entrada para la red neuronal.
5. Crear los ficheros de salida y gráficas.

A continuación se detallan cada una de las operaciones anteriores:

3.4.1 - Generar los valores de representación de cada átomo mediante funciones de simetría:

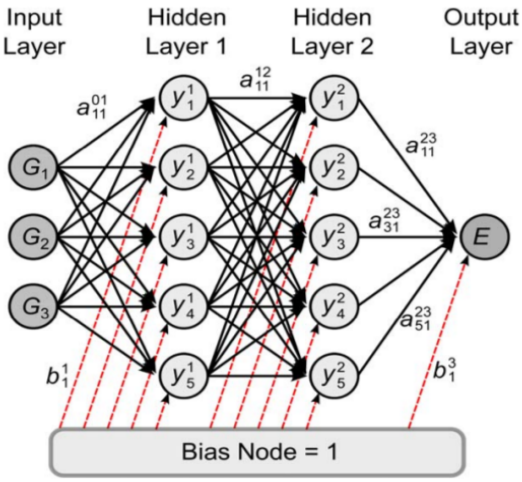


Figura 5: red neuronal.

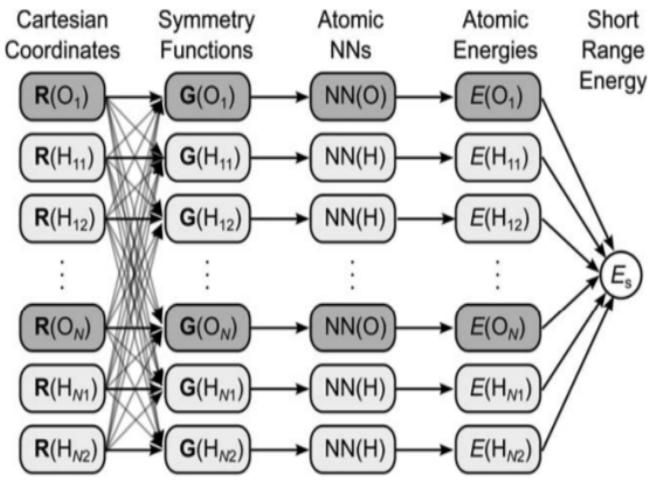


Figura 6: red neuronal de alta dimesionalidad.

La necesidad de utilizar funciones de simetría emerge de la condición de alta dimensionalidad de la red neuronal que se dispone a diseñar. Como se puede observar, en el caso de la red neuronal pequeña (figura 5) el vector de entradas $\{G_i\}$ corresponde a las coordenadas cartesianas de los átomos y sólo existe una red neuronal para cada configuración. Sin embargo, en el caso de la red neuronal de alta dimensionalidad (figura 6) el vector de entradas $\{G_i\}$ corresponde al conjunto de

valores de representación, obtenidos de evaluar las funciones de simetría, que caracterizan al átomo; existiendo una red neuronal por átomo. Por lo tanto, para crear este vector de entrada se debe realizar una operación previa con el objeto de evaluar dichas funciones de simetría para cada átomo.

Si en lugar de utilizar los valores de representación se emplearan las coordenadas atómicas como entrada directa a la red neuronal (como se hacía antes del trabajo de Behler y Parrinello[3]), ésta no sería "transferible" a sistemas con diferente número de partículas. Esta última es una propiedad esencial que se quiere que satisfaga la red neuronal a diseñar e implementar.

Por consiguiente, la operación a realizar consiste en transformar las coordenadas de un átomo en múltiples valores de representación (obtenidos de evaluar las funciones de simetría) que caracterizarán a dicho átomo dentro de un "entorno limitado esférico" (*cutoff*, figura 7). Dicho *cutoff* define el entorno energéticamente relevante del átomo y se establece como el radio de la esfera que limita el entorno.

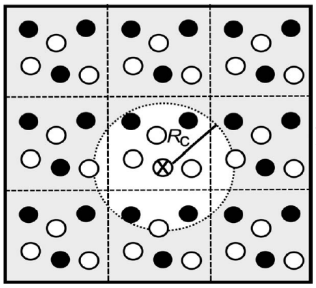


Figura 7: *cutoff* del átomo.

Para calcular dichos valores de representación se utilizarán las siguientes funciones de simetría:

1. Función de *cutoff*: se trata de un componente que utilizan todas las funciones de simetría posteriores. Esta función decrece cuando se incrementa la distancia (R_{ij}) entre el átomo central i (átomo para el que se calcula la función) y su vecino j, lo que refleja cualitativamente la reducción de las interacciones entre átomos según se distancian.

$$f_{c,1}(R_{ij}) = \begin{cases} 0.5 \cdot \left[\cos\left(\frac{\pi R_{ij}}{R_c}\right) + 1\right] & \text{for} \quad R_{ij} \leq R_c \\ 0.0 & \text{for} \quad R_{ij} > R_c \end{cases}, \qquad (1)$$

   Donde:
   R_{ij}: distancia entre el átomo central i y el átomo vecino j.
   R_c: radio del *cutoff*.

2. Función radial: esta función define el orden radial de los átomos dentro del *cutoff*, utilizando la suma de productos de las gausianas de las distancias entre átomos y la función de *cutoff*.

$$G_i^2 = \sum_{j=1}^{N_{atom}} e^{-\eta\left(R_{ij}-R_s\right)^2} \cdot f_c\left(R_{ij}\right). \qquad (2)$$

   Donde:
   η: parámetro que define el ancho de las gausianas.

R_s: parámetro que define el centro de las gausianas.

3. Función angular: esta función define el orden angular de los átomos dentro del *cutoff*, utilizando la suma sobre todos los cosenos del ángulo formado entre el átomo central i respecto a cualquier par de vecinos j y k; multiplicado por las gausianas de las tres distancias inter-atómicas del triplete de átomos y sus correspondientes funciones de *cutoff*.

$$G_i^3 = 2^{1-\zeta} \sum_{j \neq i} \sum_{k \neq i,j} \left[\left(1 + \lambda \cdot \cos\theta_{ijk}\right)^{\zeta} \cdot e^{-\eta\left(R_{ij}^2 + R_{ik}^2 + R_{jk}^2\right)} \cdot f_c\left(R_{ij}\right) \cdot f_c\left(R_{ik}\right) \cdot f_c\left(R_{jk}\right) \right] \quad (3)$$

Donde:
η: parámetro que define el ancho de las gausianas.
ζ: parámetro que define la distribución de los ángulos.
λ: parámetro que cambia la forma de la función del coseno.
θ_{ijk}: ángulo, respecto al átomo central i, que forman los tres átomos i, j y k.

Para los parámetros de ajuste mencionados anteriormente, se usarán los siguientes valores (son los definidos en el artículo de Behler[1]):

η = { 0, 0.04, 0.14, 0.32, 0.71, 1.79 }
R_s = { 0, 1, 2, 3, 4, 5 }
ζ = { 1, 2, 4, 16 }
λ = { -1, 1}
R_c = 36

Cabe señalar que el radio del *cutoff* (R_c) se define como 6Å pero se utilizará como valor de referencia R_c^2 = 36Å, con objeto de evitar el cálculo de raíces cuadradas en la determinación de distancias inter-atómicas.

Finalmente, la operación de cálculo de funciones de simetría consistirá en evaluar cada unas de las funciones anteriores para cada átomo de la configuración y para cada combinación de parámetros posible. De este modo, cada átomo quedará representado por el conjunto de valores obtenidos del cálculo anterior, que son los denominados valores de representación. Dado que este procedimiento lo permite, dicho conjunto de operaciones se realizarán en paralelo utilizando la GPU.

3.4.2 - Seleccionar las características exclusivamente necesarias:

Una vez se tengan los valores de representación de cada átomo se debe realizar una selección de características con el objetivo de reducir la dimensionalidad de los datos. Por lo tanto se debe comprobar que:

- Para el mismo átomo no existen dos valores de representación iguales, con lo que se evita la redundancia de datos.
- Para una misma función de simetría y para todos los átomos el valor de representación es

cero, con lo que se evitan funciones de simetría que generan valores de representación nulos.

3.4.3 - Comprobar que los valores de representación de los átomos caracterizan correctamente la configuración:

Tras la selección de características se debe comprobar que los valores obtenidos por las funciones de simetría representan correctamente al átomo. Para ello se deben realizar, para el conjunto valores de representación de toda la configuración de átomos, dos comprobaciones:

- Los valores de representación se deben poder diferenciar lo máximo posible: si se obtienen valores muy próximos, iguales o no distinguibles se tienen que descartar dichos valores y evaluar las funciones de simetría otra vez con parámetros nuevos.
- Para cada pareja de átomos debe existir una correlación en la magnitud entre los valores de representación en términos de funciones de simetría (que definen su entorno físico) y las fuerzas a las que están sometidos dichos átomos. Si se define la "distancia" entre dos átomos como $d_{ij} = \sqrt{\sum_{\alpha} \left(G_i^2 - G_j^2\right)^2}$, donde α designa las funciones de simetría, entonces si $d_{ij} \rightarrow 0$, $f_{ij} = \left|\vec{F}_i - \vec{F}_j\right| \rightarrow 0$. En caso contrario, la representación en términos de funciones de simetría es insuficiente y el conjunto de estas debe ser ampliado.

La única manera de realizar estas comprobaciones es implementar la relación entre d_{ij} y f_{ij}, ejecutarla, escribir los resultados en ficheros para poder representarlos en una gráfica y, finalmente, analizar dichas gráficas. A continuación se representan los datos obtenidos de la ejecución del módulo tras realizar las operaciones antes mencionadas.

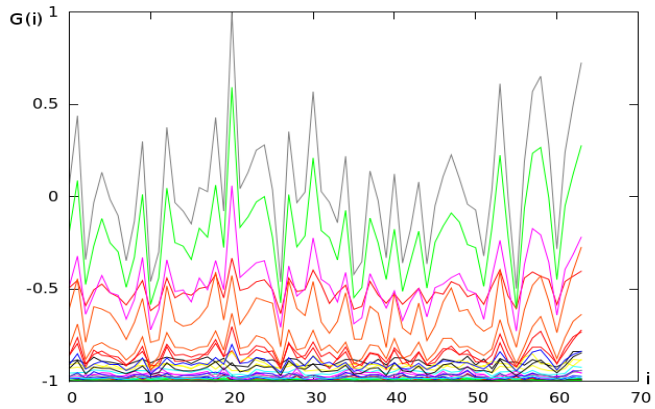


Figura 8: valores de representación normalizados.

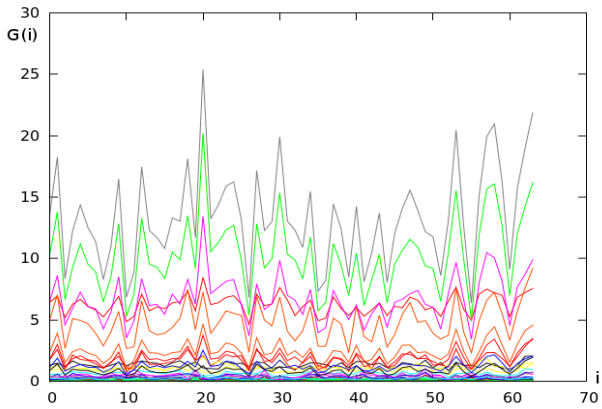


Figura 9: valores de representación sin normalizar.

En las figuras 8 y 9 (datos normalizados y sin normalizar, respectivamente) se puede observar como los valores de representación obtenidos para un conjunto (*box*) de átomos, tras evaluar las funciones de simetría, son similares pero diferenciables.

En las siguientes gráficas se representan las d_{ij} frente a f_{ij}:

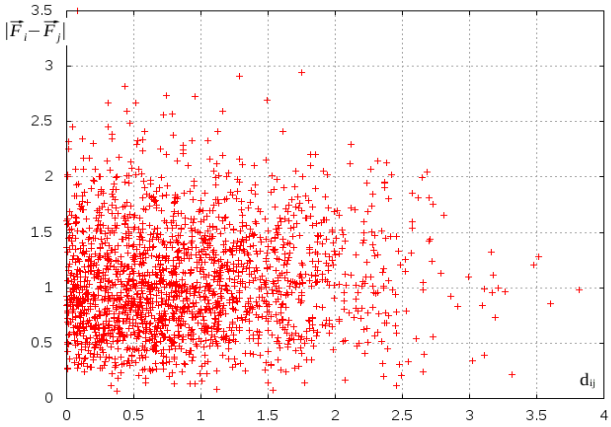


Figura 10: d_{ij} frente a $|\vec{F}_i-\vec{F}_j|$ con un único valor.

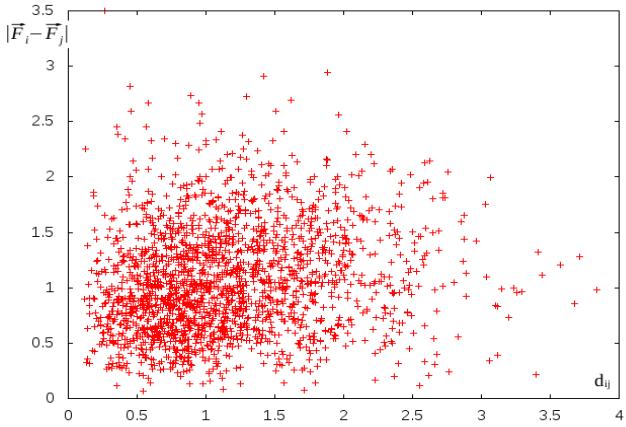


Figura 11: d_{ij} frente a $|\vec{F}_i-\vec{F}_j|$ con 15 valores.

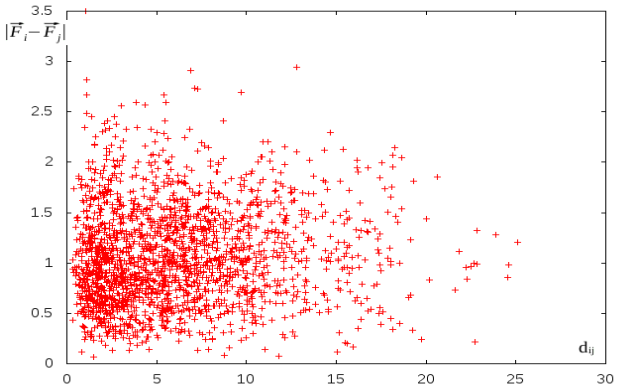


Figura 12: d_{ij} frente a $|\vec{F}_i-\vec{F}_j|$ con 30 valores (total).

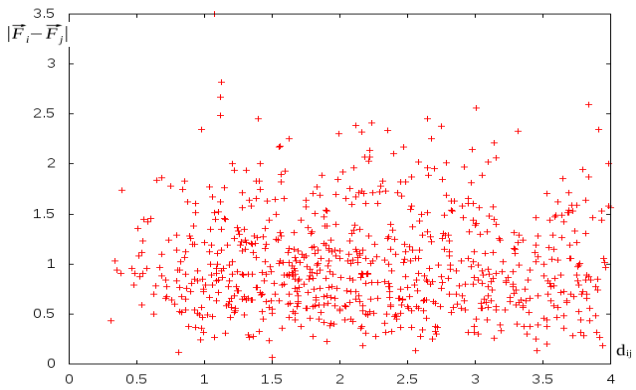


Figura 13: d_{ij} frente a $|\vec{F}_i-\vec{F}_j|$ con 30 valores (detalle).

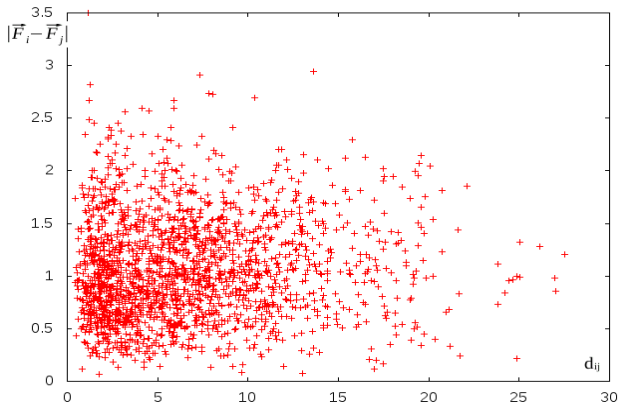


Figura 14: d_{ij} frente a $|\vec{F}_i-\vec{F}_j|$ con 57 valores (total).

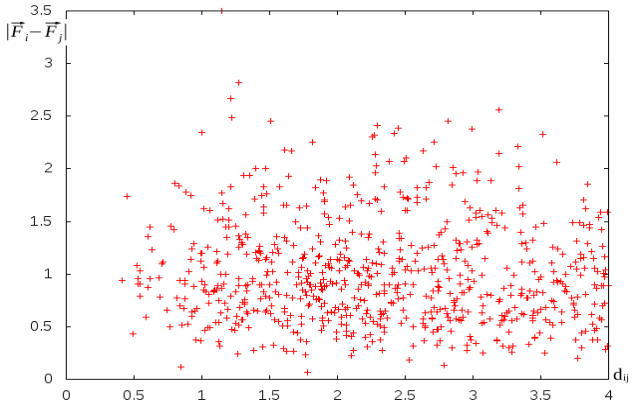


Figura 15: d_{ij} frente a $|\vec{F}_i-\vec{F}_j|$ con 57 valores (detalle).

En la figura 10 se puede observar que si se utiliza un único valor de representación, obtenido de evaluar una única función de simetría por átomo, la relación de magnitud está descompensada. Se tienen muchos valores nulos o muy cercanos al cero – el centro de la distribución se aproxima al (0.5, 1) – para d_{ij} (entornos físicos idénticos dentro de la representación) con $|\vec{F}_i-\vec{F}_j|\neq 0$, lo que quiere decir que las fuerzas a que están sometidos los átomos i y j son diferentes.

Para quince valores de representación (figura 11) podemos observar como la relación empieza a ser más equitativa y los puntos se alejan del cero y se dispersan, con un centro de distribución cercano al (1, 1). A partir de treinta valores de representación (figuras 12 y 13) se debe cambiar la escala ya que se observa una mayor dispersión de los puntos, lo que representa una relación de magnitud más equitativa separando aún más los puntos del cero.

Según el artículo de Behler[1], se requieren entre 45 y 100 valores de representación para caracterizar correctamente el entorno físico de un átomo por lo que también se representa la gráfica (figuras 14 y 15) para todas las funciones de simetría de la ejecución (un total de 57). Cabe señalar que, dado que se aplica una selección de características, el número de valores de representación puede variar; sin embargo se puede observar que a partir de 30 valores el método representa correctamente los entornos atómicos.

Es importante anotar que esta segunda comprobación sólo se ha realizado para el entorno de certificación de valores válidos cuyo resultado se muestra aquí. Sin embargo, no se incluye el diseño de esta comprobación y el código fuente estará comentado en la entrega final de la implementación.

Finalmente, las gráficas anteriores se pueden visualizar mediante los ficheros "data/G_NORMALIZED", "data/G_GRAPH" y "data/G_FORCE_RELATION_GRAPH.dat" utilizando los programas Gnuplot o Xmgrace.

3.4.4 - Normalizar los datos de entrada para la red neuronal:

Con el objetivo de representar una configuración con los mínimos errores y obtener una capacidad de ajuste de la red neuronal lo más eficiente posible, los datos se deben normalizar. Para el caso dado se utilizará el método de *ranging* con los límites [-1 , 1], por lo que se aplicará la función siguiente a todos los valores de representación (tanto de aprendizaje como de predicción):

$$G_i^{\text{scaled}} = \frac{2\left(G_i - G_{i,\min}\right)}{G_{i,\max} - G_{i,\min}} - 1, \qquad (4)$$

La decisión de diseño relativa al método de normalización (*ranging*) viene dada porque los nodos de la red neuronal utilizarán funciones de estimulación tipo sigmoide centradas en cero, y cuya respuesta estará acotada entre -1 y 1. Finalmente, todo ello facilitará el proceso de aprendizaje (optimización de los parámetros de la red neuronal).

3.4.5 - Crear los ficheros de salida y gráficas:

Finalmente, se escribirán los ficheros siguientes:

- "data/G2_SYMMETRY_LEARN.dat" → valores de representación obtenidos de evaluar la función de simetría (2) para todos los átomos de los conjuntos de aprendizaje.
- "data/G3_SYMMETRY_LEARN.dat" → valores de representación obtenidos de evaluar la función de simetría (3) para todos los átomos de los conjuntos de aprendizaje.

- "data/G2_SYMMETRY_PREDICT.dat" → valores de representación obtenidos de evaluar la función de simetría (2) para todos los átomos de los conjuntos de predicción.
- "data/G3_SYMMETRY_PREDICT.dat" → valores de representación obtenidos de evaluar la función de simetría (3) para todos los átomos de los conjuntos de predicción.
- "data/MISC_DATA.dat" → en este fichero se guardarán los siguientes datos:
  - Media de la energía de los conjuntos de aprendizaje: necesaria para el futuro ajuste de parámetros.
  - Energía máxima y mínima de los conjuntos de aprendizaje: necesarias para el futuro ajuste los parámetros.
  - Número de cajas (timesteps) de aprendizaje.
  - Número de cajas (timesteps) de predicción.
  - Número de átomos por caja.

  Este fichero es necesario para poder separar la ejecución de los módulos 1, 2 y 3 de los módulos 4, 5 y 6. Así, se consigue independizar el procesamiento de los ficheros de entrada y el pre-condicionamiento de los datos (que sólo se necesita ejecutar una vez) de la red neuronal (que se puede ejecutar varias veces con distintas configuraciones).
- "data/ENERGY_LEARN.dat": energía de todas las configuraciones de aprendizaje. Este fichero será leído por la red neuronal con el objeto de obtener el valor del error cuadrático de la energía tras cada época (error de predicción).
- "data/ENERGY_PREDICT.dat": energía de todas las configuraciones de predicción. Mediante este fichero se comprobará el error medio de predicción y se podrá observar el proceso de aprendizaje de la red neuronal.
- "data/G_GRAPH.dat": fichero de valores de representación de aprendizaje finales, tras la selección de características, sin normalizar. Utilizado para gráficas.
- "data/G_NORMALIZED_LEARN.dat": fichero de valores de representación de aprendizaje finales, tras la selección de características, normalizados. Además, en la primera linea se añade el número de valores de representación que contiene el fichero por cada átomo (valor comentado con una #). Este fichero representa la capa de entrada de la red neuronal en modo aprendizaje.
- "data/G_NORMALIZED_PREDICT.dat": fichero de valores de representación de predicción finales, tras la selección de características, normalizados. Además, en la primera linea se añade el número de valores de representación que contiene el fichero por cada átomo (valor comentado con una #). Este fichero representa la capa de entrada de la red neuronal en modo predicción.
- "data/G_FORCE_RELATION_GRAPH.dat": fichero con la relación entre la distancia d_{ij} de dos átomos y las fuerzas f_{ij} a las que están sometidos. Este fichero sólo se creará si se descomenta el código relativo a su cálculo. Utilizado para gráficas.

3.4.6 - Algoritmo:

A continuación se detalla el algoritmo del módulo completo:

Algoritmo_HDNNP v:0.4; Módulo 3 → Pre-condicionado de datos para la red neuronal
Datos de entrada → fichero "data/LEARNING_DATA.dat" y "data/PREDICT_DATA.dat".

Datos de salida → ficheros "data/G2_SYMMETRY_LEARN.dat", "data/G3_SYMMETRY_LEARN.dat", "data/G2_SYMMETRY_PREDICT.dat", "data/G3_SYMMETRY_PREDICT.dat", "data/MISC_DATA.dat", "data/ENERGY_LEARN.dat", "data/ENERGY_PREDICT.dat", "data/G_GRAPH.dat", "data/G_NORMALIZED_LEARN.dat" y "data/G_NORMALIZED_PREDICT.dat" [Opcional: "data/G_FORCE_RELATION_GRAPH.dat"].

comienzo_algoritmo
[Fase 1 → Funciones de simetría y selección de características]
Generar las combinaciones para la función de simetría (2) con los parámetros η y R_s
Generar las combinaciones para la función de simetría (3) con los parámetros η, ζ y λ
[Funciones de simetría conjuntos de aprendizaje]
Abrir el fichero de entrada "data/LEARNING_DATA.dat" en modo lectura
Abrir los ficheros de salida "data/G2_SYMMETRY_LEARN.dat" y "data/G3_SYMMETRY_LEARN.dat" en modo escritura
Escribir en los ficheros de salida "data/G2_SYMMETRY_LEARN.dat" y "data/G3_SYMMETRY_LEARN.dat" las cabeceras indicando la combinación seleccionada para cada función de simetría
Abrir el fichero de salida "data/ENERGY_LEARN.dat" en modo escritura
Mientras los ficheros de entrada contengan datos, procesar una configuración
Leer del fichero de entrada "data/LEARNING_DATA.dat" la energía y escribirla en el fichero de salida "data/ENERGY_LEARN.dat". Almacenar la energía en el acumulador para calcular la media y guardar valor máximo y mínimo.
Leer del fichero de entrada "data/LEARNING_DATA.dat" el tamaño de la caja
Para todos los átomos de la configuración
Leer del fichero de entrada "data/LEARNING_DATA.dat" las coordenadas del átomo
Leer del fichero de entrada "data/LEARNING_DATA.dat" las fuerzas del átomo
Calcular los valores de representación de cada átomo evaluando las funciones de simetría, utilizando la GPU
Escribir en el fichero de salida "data/G2_SYMMETRY_LEARN.dat" los valores de representación obtenidos de evaluar la función de simetría (2) para cada átomo y almacenar valor de representación mínimo y máximo (para cada función de simetría).
Escribir en el fichero de salida "data/G3_SYMMETRY_LEARN.dat" los valores de representación obtenidos de evaluar la función de simetría (3) para cada átomo y almacenar valor de representación mínimo y máximo (para cada función de simetría).
[Opcional: Calcular para cada pareja de átomos la relación valores de representación respecto de sus fuerzas. Fichero "data/G_FORCE_RELATION_GRAPH.dat"]
Cerrar fichero de entrada "data/LEARNING_DATA.dat" y de salida "data/ENERGY_LEARN.dat", "data/G2_SYMMETRY_LEARN.dat" y "data/G3_SYMMETRY_LEARN.dat"
[Funciones de simetría conjuntos de predicción]
Abrir el fichero de entrada "data/PREDICT_DATA.dat" en modo lectura
Abrir los ficheros de salida "data/G2_SYMMETRY_PREDICT.dat" y

"data/G3_SYMMETRY_PREDICT.dat" en modo escritura
Escribir en los ficheros de salida "data/G2_SYMMETRY_PREDICT.dat" y "data/G3_SYMMETRY_PREDICT.dat" las cabeceras indicando la combinación seleccionada para cada función de simetría
Abrir el fichero de salida "data/ENERGY_PREDICT.dat" en modo escritura
Mientras los ficheros de entrada contengan datos, procesar una configuración
    Leer del fichero de entrada "data/PREDICT_DATA.dat" la energía y escribirla en el fichero de salida "data/ENERGY_PREDICT.dat".
    Leer del fichero de entrada "data/PREDICT_DATA.dat" el tamaño de la caja
    Para todos los átomos de la configuración
        Leer del fichero de entrada "data/PREDICT_DATA.dat" las coordenadas del átomo
        Leer del fichero de entrada "data/PREDICT_DATA.dat" las fuerzas del átomo
        Calcular los valores de representación de cada átomo evaluando las funciones de simetría, utilizando la GPU
        Escribir en el fichero de salida "data/G2_SYMMETRY_PREDICT.dat" los valores de representación obtenidos de evaluar la función de simetría (2) para cada átomo y almacenar valor mínimo y máximo (para cada función de simetría).
        Escribir en el fichero de salida "data/G3_SYMMETRY_PREDICT.dat" los valores de representación obtenidos de evaluar la función de simetría (3) para cada átomo y almacenar valor mínimo y máximo (para cada función de simetría).
Cerrar fichero de entrada "data/PREDICT_DATA.dat" y de salida "data/ENERGY_PREDICT.dat", "data/G2_SYMMETRY_PREDICT.dat" y "data/G3_SYMMETRY_PREDICT.dat"
[Selección de características]
Selección de características: eliminar/marcar los valores de representación con valor nulo para la misma función de simetría en todos los átomos.
Selección de características: eliminar/marcar los valores de representación con mismo valor para distinta función de simetría en todos los átomos.
Abrir fichero de salida "data/MISC_DATA.dat"
Escribir en el fichero de salida "data/MISC_DATA.dat" la media de la energía de aprendizaje.
Escribir en el fichero de salida "data/MISC_DATA.dat" el valor mínimo y máximo de la energía de aprendizaje.
Escribir en el fichero de salida "data/MISC_DATA.dat" el número de cajas (*timesteps*) de aprendizaje.
Escribir en el fichero de salida "data/MISC_DATA.dat" el número de cajas (*timesteps*) de predicción.
Escribir en el fichero de salida "data/MISC_DATA.dat" el número de átomos por caja.
Cerrar el fichero de salida "data/MISC_DATA.dat"
[Fase 2 → Normalización y escritura de ficheros de salida]
[Conjuntos de aprendizaje]
Abrir los ficheros de entrada "data/G2_SYMMETRY_LEARN.dat" y "data/G3_SYMMETRY_LEARN.dat" en modo lectura

Saltar la primera línea de los ficheros de entrada “data/G2_SYMMETRY_LEARN.dat” y “data/G3_SYMMETRY_LEARN.dat”
Abrir los ficheros de salida “data/G_NORMALIZED_LEARN.dat” y “data/G_GRAPH.dat” en modo escritura
Escribir en el fichero de salida “data/G_NORMALIZED_LEARN.dat” el número de valores de representación que contiene cada átomo (comentado con una #)
Mientras los ficheros de entrada contengan datos
Para todos los átomos
Leer los valores de representación del fichero “data/G2_SYMMETRY_LEARN.dat” y escribirlos (sólo los válidos) en los ficheros “data/G_GRAPH.dat” (sin normalizar) y “data/G_NORMALIZED_LEARN.dat” (normalizados)
Leer los valores de representación del fichero “data/G3_SYMMETRY_LEARN.dat” y escribirlos (sólo los válidos) en los ficheros “data/G_GRAPH.dat” (sin normalizar) y “data/G_NORMALIZED_LEARN.dat” (normalizados)
Cerrar los ficheros de entrada y de salida
[Conjuntos de predicción]
Abrir los ficheros de entrada “data/G2_SYMMETRY_PREDICT.dat” y “data/G3_SYMMETRY_PREDICT.dat” en modo lectura
Saltar la primera línea de los ficheros de entrada “data/G2_SYMMETRY_PREDICT.dat” y “data/G3_SYMMETRY_PREDICT.dat”
Abrir el fichero de salida “data/G_NORMALIZED_PREDICT.dat” en modo escritura
Escribir en el fichero de salida “data/G_NORMALIZED_PREDICT.dat” el número de valores de representación que contiene cada átomo (comentado con una #)
Mientras los ficheros de entrada contengan datos
Para todos los átomos
Leer los valores de representación del fichero data/G2_SYMMETRY_PREDICT.dat” y escribirlos (sólo los válidos) en el fichero “data/G_NORMALIZED_PREDICT.dat” (normalizados)
Leer los valores de representación del fichero “data/G3_SYMMETRY_PREDICT.dat” y escribirlos (sólo los válidos) en el fichero “data/G_NORMALIZED_PREDICT.dat” (normalizados)
Cerrar los ficheros de entrada y de salida
fin_algoritmo

A continuación, se procede a descomponer el algoritmo del módulo para desarrollar el cálculo de los valores de representación de cada átomo, utilizando la GPU. Debido a que se va a paralelizar utilizando la GPU se expone el diseño utilizado para la configuración de bloques e hilos.

3.4.6.1 – ***Calcular los valores de representación de cada átomo evaluando las funciones de simetría, utilizando la GPU:***

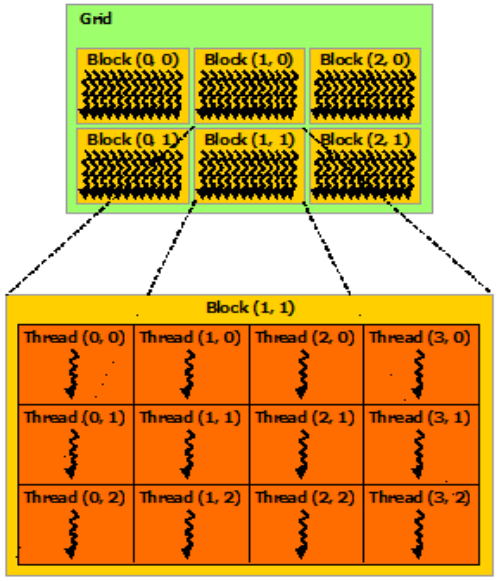


Figura 16: arquitectura lógica GBT (Grid-Block-Thread) en CUDA.

Dentro de la arquitectura CUDA se deben configurar las llamadas al código que se ejecutará en la GPU (*kernel*), definiendo el número de bloques y de hilos a utilizar, dentro de las propias invocaciones. Para el diseño a realizar se ha decidido asignar los bloques a los átomos centrales i (sobre los que se calcula las funciones de simetría) y los hilos a los átomos vecinos de primera instancia j (sobre los que se pivota en primera instancia). Por lo tanto, esta paralelización ahorra dos bucles: el de recorrer los átomos centrales y el de pivotar sobre los vecinos de primera instancia. Para los vecinos de segunda instancia k (necesarios para la función de simetría angular (3)) se necesitará un bucle dentro de la GPU. Finalmente, la llamada al *kernel* resultará en la siguiente forma:

```
kernel<<<número_de_átomos, número_de_átomos>>>(parámetro1, parámetro2,
…,parámetro N);
```

Algoritmo_HDNNP v:0.4.1; Módulo 3 → calculo de los valores de representación de cada átomo evaluando las funciones de simetría. Versión GPU.

comienzo_algoritmo
    Iniciación y copia de la CPU a la GPU de las estructuras relativas a las combinaciones de las funciones de simetría (2)
    Iniciación y copia de la CPU a la GPU de las estructuras relativas a las combinaciones de las funciones de simetría (3)
    Iniciación y copia de la CPU a la GPU de las estructuras relativas a la configuración de átomos
    Iniciación y copia de la CPU a la GPU de la dimensión de la caja
    Llamada al *kernel GPU*
    Copia de la GPU a la CPU de la estructuras relativas a la configuración de átomos (dentro de estas estructuras se encuentran los valores de representación calculados en la GPU)
fin_algoritmo

A continuación se descompone el módulo de llamada al *kernel_GPU*:

Algoritmo_HDNNP v:0.4.2; Módulo 3 → llamada al kernel_GPU.

comienzo_algoritmo
El índice de bloque (blockIdx.x) corresponde al átomo central
El índice de hilo (threadIdx.x) corresponde al átomo vecino en primera instancia
Se define un acumulador en memoria compartida (para los sumatorios de las funciones de simetría (2) y (3))
Para cada combinación de la función de simetría (2)
Se inicializa el acumulador (sólo un hilo, necesaria sincronización de hilos) a cero
Si los átomos no son iguales (blockIdx.x != threadIdx.x)
Calcular distancia entre átomos (R_{ij})
Si los átomos están dentro del *cutoff* (R_{ij} <= R_c)
Se suma al acumulador la evaluación de la función de simetría (2)
para este par de átomos {i, j} (operación atómica)
Sincronizar hilos
Almacenar en el átomo central para la combinación actual el valor del acumulador
Para cada combinación de la función de simetría (3)
Se inicializa el acumulador (sólo un hilo, necesaria sincronización de hilos) a cero
Si los átomos no son iguales (blockIdx.x != threadIdx.x)
Calcular distancia entre átomos (R_{ij})
Si los átomos están dentro del *cutoff* (R_{ij} <= R_c)
Para cada átomo de la configuración
Si los átomos no son iguales (blockIdx.x != threadIdx.x != vecino_2ª_instancia)
Calcular distancia entre el átomo central y el vecino de segunda instancia (R_{ik})
Si los átomos están dentro del *cutoff* (R_{ik} <= R_c)
Calcular distancia entre el átomo vecino de primera instancia y el vecino de segunda instancia (R_{jk})
Se suma al acumulador la evaluación de la función de simetría (3) para este trío de átomos {i, j, k} (operación atómica)
Sincronizar hilos
Almacenar en el átomo central para la combinación actual el valor del acumulador
fin_algoritmo

3.4.7 - Reseñas:

1. Se utiliza un acumulador en memoria compartida con el objetivo de maximizar el uso de los registros de la GPU y minimizar los acceso a la memoria global de la GPU. Mediante esta técnica se optimiza el acceso a memoria aunque los hilos tengan que cooperar entre ellos. Por lo tanto, sólo se accede a la memoria global en el momento de almacenar los acumuladores de cada átomo para cada combinación de las funciones de simetría.
2. El cálculo de distancias entre átomos se debe realizar teniendo en cuenta la periodicidad de

la caja.

3. En la evaluación de la función de simetría (3), el cálculo del coseno entre los tres átomos se realiza mediante la regla del coseno dado que se tienen las distancias entre átomos.
4. Todas las llamadas a funciones matemáticas dentro de la GPU se realizan con versiones "intrinsics" (optimizadas para el hardware de NVIDIA), con objeto de optimizar el rendimiento de la GPU.
5. Se debe minimizar el uso de la operación raíz cuadrada por lo que el radio del *cutoff* (R_c) se utiliza en su forma cuadrática R_c^2 para determinar qué átomos se encuentran dentro de la zona energéticamente significativa . Por lo tanto, la raíz cuadrada, sólo es necesaria dentro del cálculo del ángulo que forman los tres átomos para evaluar la función de simetría (3) (regla del coseno).

3.4.8 - Posibles mejoras:

Si se dispone de GPUs con varios procesadores (multi-GPU) se puede paralelizar las llamadas a los *kernels*: de forma que a cada *grid* se le asigna el átomo central i, a cada bloque el átomo vecino de primera instancia j y a cada hilo el átomo vecino de segunda instancia k. De esta forma se elimina el bucle necesario para recorrer los vecinos de segunda instancia k, optimizando la paralelización.

Módulo 4 → Ejecución de la red neuronal (modo aprendizaje):

El objetivo de este módulo es ejecutar la red neuronal con todos los conjuntos de átomos de aprendizaje, obteniendo un ajuste óptimo de los parámetros de dicha red. Para realizar esta labor se debe:

1. Leer los valores de representación normalizados de cada átomo para una configuración dada (del conjunto de aprendizaje), inicializar una red neuronal por átomo, obtener la predicción atómica de dicha red neuronal y, finalmente, sumar todas las predicciones atómicas resultantes de las redes neuronales obteniendo la predicción de la configuración (caja).
2. Calcular el error de predicción, pudiéndose dar dos casos:
    1. Si el error supera un determinado umbral, realizar una optimización de los parámetros (comunes a todas las redes neuronales atómicas) y volver al paso 1 con el mismo conjunto de átomos.
    2. Si el error se encuentra dentro del umbral o el método converge se debe volver al punto 1 con un nuevo conjunto de átomos.

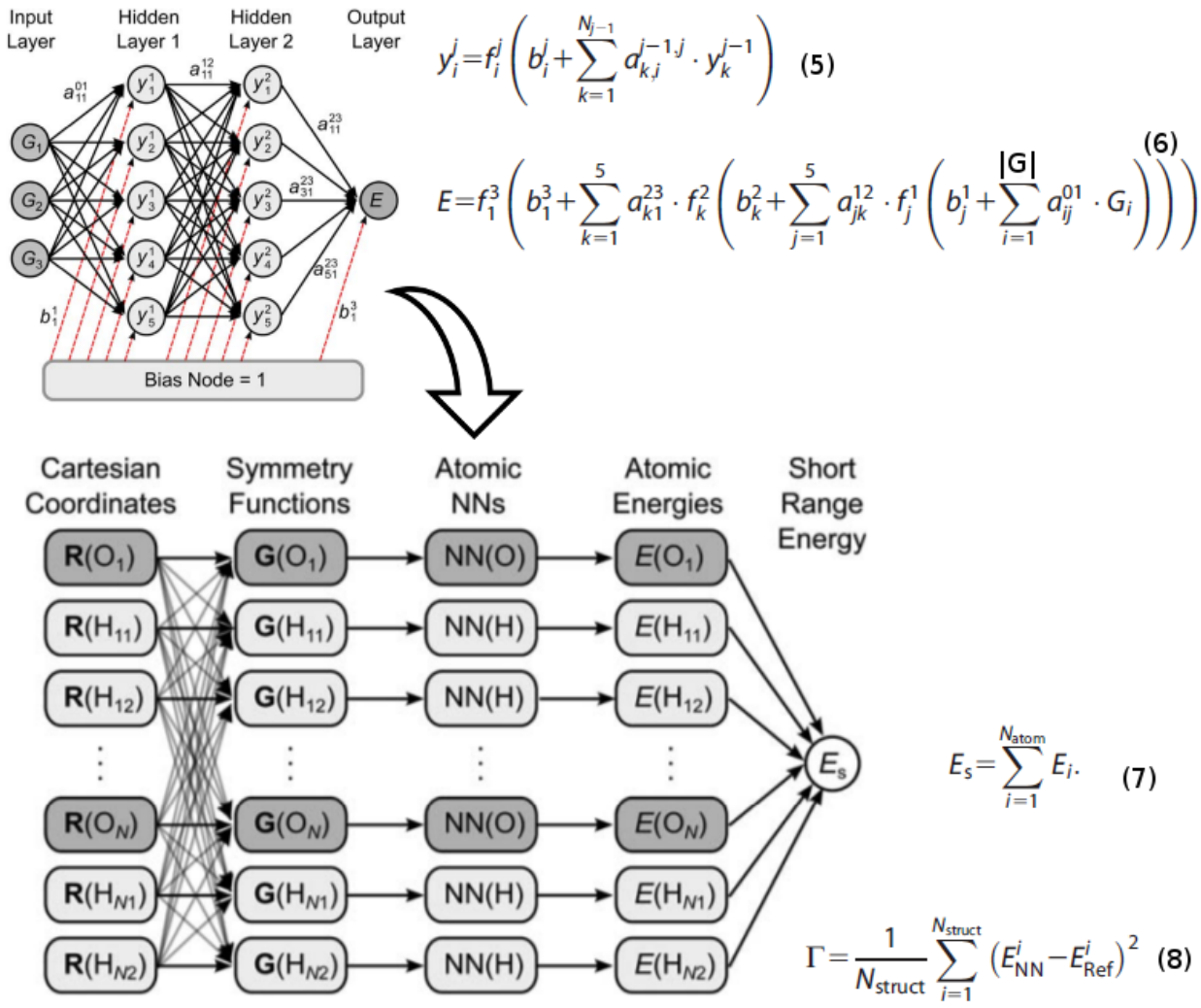
Por lo tanto, se definen dos operaciones diferentes:

- Época de la red neuronal: leer los datos de representación normalizados, inicializar la red neuronal, calcular la predicción y el error de predicción.
- Optimización: optimizar los parámetros de la red neuronal de modo que se minimice el error de predicción.

A continuación se detallan las dos operaciones:

Antonio Díaz Pozuelo – adpozuelo@uoc.edu

3.5.1 - Época de la red neuronal:



$$y_i^j = f_i^j\left(b_i^j + \sum_{k=1}^{N_{j-1}} a_{k,i}^{j-1,j} \cdot y_k^{j-1}\right) \quad (5)$$

$$E = f_1^3\left(b_1^3 + \sum_{k=1}^{5} a_{k1}^{23} \cdot f_k^2\left(b_k^2 + \sum_{j=1}^{5} a_{jk}^{12} \cdot f_j^1\left(b_j^1 + \sum_{i=1}^{|G|} a_{ij}^{01} \cdot G_i\right)\right)\right) \quad (6)$$

$$E_s = \sum_{i=1}^{N_{atom}} E_i. \quad (7)$$

$$\Gamma = \frac{1}{N_{struct}} \sum_{i=1}^{N_{struct}} \left(E_{NN}^i - E_{Ref}^i\right)^2 \quad (8)$$

Figura 17: época de la red neuronal de alta dimensionalidad.

En la figura 17 se puede observar el diseño de la red neuronal de alta dimensionalidad y la formulación matemática para la predicción de la energía del sistema. Concretamente, se puede comprobar como cada átomo inicializa una red neuronal atómica cuya entrada serán los valores de representación del átomo. Dentro de la red neuronal atómica se tendrán:

- G_i valores de representación del átomo como capa de entrada (*input*)
- Dos capas ocultas j.
- Cinco neuronas por capa: y^j_k e y^j_k, siendo k el número de la neurona dentro de la capa.
- Una neurona de salida (*output*): E.
- N parámetros: $a^{j-1}{}_k{}^j{}_i$, siendo i el número del parámetro dentro la capa j que proviene de la neurona k; y b^j_i, siendo i el el parámetro que va desde el nodo bias a la neurona i de la capa j.

Todos los parámetros serán comunes a todas las redes neuronales atómicas y su número dependerá de la cantidad de valores de representación (|G|). Dado que la selección de características en el pre-condicionamiento de los datos (módulo previo) se realiza en tiempo de ejecución, se tendrá el siguiente número de parámetros: |a| + |b| = (|G| * $|y^1|$) + ($|y^1|$ * $|y^2|$) + $|y^2|$ + $|y^1|$ + $|y^2|$ + |E|. Finalmente, todos los parámetros serán inicializados con valores aleatorios en el rango [-1, 1].

- Una función de estimulación neuronal: $y^j_i = f^j_i$ que se evaluará para determinar el valor de la neurona y_i en la capa j. Por un lado, para las dos capas ocultas j la función de estimulación neuronal utilizada será la sigmoide $y = \frac{1}{1+e^{-x}}$ dado que se necesita una función no lineal (cabe recordar que la normalización de los valores de representación y la inicialización de los parámetros se han realizado en el rango [-1, 1], dado que es la imagen de la función sigmoide). Por otro lado, para la neurona de salida E se utilizará una función lineal, concretamente la suma de todos los parámetros multiplicados por los valores de las neuronas de la capa anterior.

Una vez que todas las redes neuronales han calculado su estimación de energía para cada átomo, éstas se deben sumar para obtener la predicción del sistema E_s. Este valor será utilizado para calcular el error de la predicción total como:

$$\Gamma = \frac{1}{N_{\text{struct}}} \sum_{i=1}^{N_{\text{struct}}} \left(E^i_{\text{NN}} - E^i_{\text{Ref}}\right)^2 \quad (8)$$

Siendo el número de estructuras igual al número de átomos, E^i_{NN} igual a E_s y E^i_{Ref} la energía de referencia que ya se posee (supervisada).

Finalmente, el diseño de la ejecución de épocas de la red neuronal se realiza para su ejecución en paralelo utilizando la GPU.

3.5.2 – Optimización o aprendizaje:

El proceso de optimización o aprendizaje se basa en ajustar, tras la ejecución de cada época, el valor de los N parámetros $a^{j-1}_k{}^j_i$ de la red neuronal con el objetivo de minimizar el error de predicción.

La idea original para la optimización de los parámetros era diseñar e implementar un filtro digital extendido de Kalman, tal y como se expone en el artículo de Witkoskie y Doren[2]. Sin embargo, y tras realizar el diseño e implementación de ejemplo del oscilador armónico que indica el artículo, no se ha podido utilizar dicho filtro ya que el método no convergía. Investigando el motivo de la no convergencia se ha encontrado una incongruencia en la página 16 de dicho artículo[2].

Concretamente, la función T4 (*table 1: Equations of the Extended Kalman Filter*) que configura las matrices R_k y Q_k no concuerda con la explicación que se da sobre ella más adelante: "[...] *The error vector f(x)-f˜(x;θ) is used to calculate the update for R_k, while H_k (f(x)-f˜(x;θ)) is used in the update for Q_k* [...]".

Con objeto de ahondar más en el método se realizó un estudio independiente del filtro de Kalman (y

de su versión extendida) utilizando las referencias encontradas en los artículos de Behler[1] y de Witkoskie y Doren[2]. Sin embargo, dado que el material es muy amplio y complejo y la fecha límite de entrega del hito se aproximaba se decidió abandonar esta vía.

Queriendo mantener la esencia del trabajo de Behler[1] respecto a la optimización de los parámetros, se optó por utilizar una implementación en código abierto ya realizada (librería) del filtro extendido de Kalman. Tras realizar intensas búsquedas en Internet e implementar algunos ejemplos, utilizando librerías externas, se llegó a las siguientes conclusiones:

1. Las implementaciones existentes suelen ser proyectos finales de estudios universitarios, doctorales, post-doctorales, etc. Por consiguiente, las librerías son poco o nada flexibles con lo que no se pueden integrar dado que: o no permiten integración externa o la implementación del filtro se ajusta a un problema de naturaleza totalmente distinta.
2. Dichas librerías suelen ser antiguas (o muy antiguas) y carecen de soporte, documentación, etc.

Por lo tanto, también se desestimó la opción de implementar librerías de código abierto del filtro extendido de Kalman. Ejemplos de las librerías estudiadas son: https://github.com/jeremyfix/easykf y http://kalman.sourceforge.net/.

Una vez descartado el filtro de Kalman se realizó un estudio de los siguientes algoritmos de optimización:

1. SIMPLEX: se realizó una implementación de prueba del algoritmo utilizando la librería de código abierto Nlopt. Tras la prueba se comprobó que la integración de dicha librería era compleja y que el método SIMPLEX no soportaba un alto número de parámetros a optimizar.
2. COMPLEX: se realizó una implementación de prueba del algoritmo utilizando la librería privativa IMSL (compilada en FORTRAN) disponible en el centro de cálculo HPC (*High-Performance Computing*) Ladon-Hidra del Instituto de Química Física Rocasolano (CSIC). Tras la prueba se comprobó que el resultado de la optimización era satisfactorio y que la librería se podía integrar en la aplicación mediante el uso de un adaptador (*wrapper*). Dicho adaptador sirve de enlace entre los dos lenguajes: CUDA_C/C++ y FORTRAN.

Finalmente, y tras todos los estudios y pruebas citados anteriormente, se decide utilizar el optimizador COMPLEX-BCPOL de la librería IMSL aunque conlleve el diseño e implementación de un adaptador en el lenguaje FORTRAN para comunicar dicho lenguaje con CUDA_C/C++.

La ejecución de la optimización de los parámetros se realizará de forma secuencial mediante la librería IMSL y de forma paralela en el cálculo del error de predicción. Esto es debido a que el optimizador llamará a la función de coste (época de la red neuronal en modo aprendizaje), tras cada ajuste de parámetros, con el objeto de comprobar la calidad del ajuste. Por lo tanto, con objeto de paralelizar el máximo posible de operaciones, se paralelizará el cálculo de la función de coste para que devuelva el error de predicción de un número determinado (NUMBER_OF_BOXES_TO_OPT) de configuraciones de átomos (cajas).

3.5.3 - Algoritmo:

A continuación se detalla el algoritmo del módulo completo:

Algoritmo_HDNNP v:0.5; Módulo 4 → Ejecución de la red neuronal (modo aprendizaje)
Datos de entrada → ficheros "data/MISC_DATA.dat", "data/ENERGY_LEARN.dat" y "data/G_NORMALIZED_LEARN.dat".
Datos de salida → N parámetros $a^{j-1}{}_{k}{}^{j}{}_{i}$

comienzo_algoritmo
Abrir el fichero de entrada "data/G_NORMALIZED_LEARN.dat" en modo lectura
Leer del fichero "data/G_NORMALIZED_LEARN.dat" el tamaño de la capa de entrada
Abrir el fichero de entrada "data/MISC_DATA.dat" en modo lectura
Leer del fichero "data/MISC_DATA.dat" el valor máximo, mínimo y medio de la energía
Leer del fichero "data/MISC_DATA.dat" el número de átomos por caja y el número de cajas de aprendizaje
Cerrar el fichero "data/MISC_DATA.dat
Inicializar de forma aleatoria entre [-1, 1] el juego de parámetros de la red neuronal y establecer sus límites inferior y superior a {-1, 1}
Al parámetro que corresponde al BIAS → OUPUT_NEURON asignarle el valor medio de la energía y como límites inferior y superior {valor_mínimo_energía, valor_máximo_energía}
Ajustar los parámetros de configuración del optimizador (tolerancia, número máximo de llamadas a la funcion de coste, etc.)
Abrir el fichero de entrada "data/ENERGY.dat" en modo lectura
Abrir el fichero de salida "data/LEARNING_PROCESS" en modo escritura
Mientras los ficheros de entrada contengan datos
Leer NUMBER_OF_BOXES_TO_OPT configuraciones de átomos mediante sus valores de representación del fichero de entrada "data/G_NORMALIZED_LEARN.dat"
Leer NUMBER_OF_BOXES_TO_OPT energías E_{Ref} (supervisada) del fichero de entrada "data/ENERGY_LEARN.dat"
Ejecutar la época de la red neuronal (en modo aprendizaje), utilizando la GPU [llamada mediante el adaptador]
Escribir en el fichero de salida "data/LEARNING_PROCESS" el conjunto de energías predichas y las supervisadas.
Cerrar los ficheros de entrada
fin_algoritmo

3.5.3.1 - *Ejecutar la época de la red neuronal (en modo aprendizaje), utilizando la GPU [llamada mediante el adaptador]*

A continuación se detalla el algoritmo de ejecución de épocas en la red neuronal en modo aprendizaje. Cabe señalar que este algoritmo (el desarrollo dentro de la GPU) es común para la ejecución de las épocas de la red neuronal en modo predicción.

Algoritmo_HDNNP v:0.5.1; Módulo 4 → ejecutar la época de la red neuronal. Versión GPU.

comienzo_algoritmo
Iniciación y copia de la CPU a la GPU de las estructuras relativas a la representación de los átomos
Iniciación y copia de la CPU a la GPU de las estructuras relativas a la representación de las energías
Iniciación y copia de la CPU a la GPU de las estructuras relativas a los parámetros de la red neuronal (los mismos para todas las redes neuronales atómicas)
Llamada al *kernel_GPU*
Copia de la GPU a la CPU de la estructuras relativas a la representación de las energías
Si la red neuronal está en modo aprendizaje
Devolver el error de predicción $\Gamma = \frac{1}{N_{struct}} \sum_{i=1}^{N_{struct}} \left(E^i_{NN} - E^i_{Ref}\right)^2$, siendo N_{struct} igual a
NUMBER_OF_BOXES_TO_OPT
Si la red esta en modo predicción
Devolver las NUMBER_OF_BOXES_TO_OPT energías predichas
fin_algoritmo

La configuración de bloques e hilos para la llamada al *kernel* se realiza de la siguiente manera: se crearán NUMBER_OF_BOXES_TO_OPT bloques que representarán las cajas de átomos; y se crearán tantos hilos como átomos se tenga por caja por lo que cada hilo representará una red neuronal atómica. Por lo tanto, la llamada al *kernel* resultará en la siguiente forma:

```
kernel<<<NUMBER_OF_BOXES_TO_OPT, número_átomos_por_caja>>>(parámetro1,
parámetro2, …,parámetro N);
```

A continuación se descompone el módulo de llamada al *kernel_GPU*:

Algoritmo_HDNNP v:0.5.2; Módulo 4 → llamada al kernel_GPU.

comienzo_algoritmo
El índice de bloque (blockIdx.x) corresponde a la caja de átomos
El índice de hilo (threadIdx.x) corresponde al átomo (red neuronal atómica)
Se define un vector en memoria compartida para que cada átomo almacene su energía atómica
Cada átomo (hilo) obtiene su capa de entrada del vector de valores de representación
[Operaciones entre capa de entrada y capa oculta 1]
Cada átomo crea e inicializa su primera capa oculta de neuronas
Cada átomo calcula el valor temporal de cada neurona de la primera capa oculta como el sumatorio de cada elemento de la capa de entrada por el valor del parámetro que une dicho elemento con la neurona $\sum_{k=1}^{N_{j-1}} d_{k,j}^{j-1,j} \cdot Gi$
Cada átomo calcula el valor final de cada neurona de la primera capa como la estimulación (función sigmoide) de su valor más el valor del parámetro BIAS que les une.

$$y_i^j = f_i^j\left(b_i^j + \sum_{k=1}^{N_{j-1}} d_{k,i}^{j-1,j} \cdot \mathrm{Gi}\right)$$ (estimulación neuronal)

[Operaciones entre capa oculta 1 y capa oculta 2]
Cada átomo crea e inicializa su segunda capa oculta de neuronas
Cada átomo calcula el valor temporal de cada neurona de la segunda capa oculta como el sumatorio de cada neurona de la primera capa oculta por el valor del parámetro que une ambas neuronas $\sum_{k=1}^{N_{j-1}} d_{k,i}^{j-1,j} \cdot y_k^{j-1}$
Cada átomo calcula el valor final de cada neurona de la segunda capa como la estimulación (función sigmoide) de su valor más el valor del parámetro BIAS que les une.

$$f_k^2\left(b_k^2 + \sum_{j=1}^{5} a_{jk}^{12} \cdot f_j^1\left(b_j^1 + \sum_{i=1}^{|G|} a_{ij}^{01} \cdot G_i\right)\right)$$ (estimulación neuronal)

[Operaciones entre capa oculta 2 y la neurona de salida]
Cada átomo crea e inicializa su neurona de salida
Cada átomo calcula el valor temporal de su neurona de salida como el sumatorio de cada neurona de la segunda capa oculta por el valor del parámetro que une ambas neuronas
$\sum_{k=1}^{N_{j-1}} d_{k,i}^{j-1,j} \cdot y_k^{j-1}$
Cada átomo calcula el valor final de su neurona de salida como su valor más el valor del parámetro BIAS que les une.

$$E = f_1^3\left(b_1^3 + \sum_{k=1}^{5} a_{k1}^{23} \cdot f_k^2\left(b_k^2 + \sum_{j=1}^{5} a_{jk}^{12} \cdot f_j^1\left(b_j^1 + \sum_{i=1}^{|G|} a_{ij}^{01} \cdot G_i\right)\right)\right)$$

Cada átomo guarda en el vector de energías atómicas (memoria compartida) el valor E calculado
Sincronización de hilos
Se realiza una reducción binaria del vector de energías atómicas para obtener la energía de la caja
Si el átomo es el primero (tjd==0) dicho átomo guarda en el vector de energías de cajas el valor de la energía de la caja (energía predicha)
fin_algoritmo

3.5.4 – Reseñas:

1. Se utiliza un vector en memoria compartida con el objetivo de maximizar el uso de los registros de la GPU y minimizar los acceso a la memoria global de la GPU. Mediante esta técnica se optimiza el acceso a memoria aunque los hilos tengan que cooperar entre ellos.
2. Todas las llamadas a funciones matemáticas dentro de la GPU se realizan con sus versiones

"intrinsics" con objeto de optimizar el rendimiento de la GPU.

3. Se realiza una reducción binaria del vector de energías atómicas dado que su coste computacional es logarítmico y además se paraleliza la operación (cooperación entre hilos).
4. El parámetro BIAS → OUPUT_NEURON se inicializa como el valor medio de la energía y sus límites inferior y superior como {valor_mínimo_energía, valor_máximo_energía} dado que es una de las recomendaciones de Behler[1]. De esta forma el proceso de optimización está orientado desde el principio y se evita el estancamiento del mismo en mínimos locales.
5. Para todos los demás parámetros el valor de inicialización es un número aleatorio entre -1 y 1, y sus límites inferior y superior son {-1, 1}. Por lo tanto, todos los parámetros menos el BIAS → OUPUT_NEURON están restringidos dentro de los valores de la imagen de la función sigmoide (función de estimulación neuronal).
6. El vector de valores de representación de los átomos se genera tras serializar en la CPU todos los valores de representación de cada átomo de cada caja. Por consiguiente, se debe utilizar un índice de vector del tipo base más desplazamiento y realizar la operación de des-serializado en la GPU. De este modo se maximiza la propiedad de la memoria llamada coalescencia con lo que se obtiene un mayor rendimiento.

3.5.5 – Posibles mejoras:

Por un lado, el uso de un filtro digital extendido de Kalman mejoraría la optimización de parámetros ya que se trata de un método dirigido y con memoria. Por otro lado, el uso de gradientes y de las fuerzas en el proceso de optimización también mejoraría notablemente el rendimiento de la operación.

3.6 - Módulo 5 → Ejecución de la red neuronal (modo predicción):

El objetivo de este módulo es ejecutar la red neuronal, utilizando los parámetros ya optimizados tras el aprendizaje, con todos los conjuntos de átomos de predicción. De este modo, se obtiene la predicción de la energía de cada secuencia temporal atómica. Para realizar esta labor se debe:

1. Leer los valores de representación normalizados de cada átomo para una configuración dada (del conjunto de predicción), inicializar una red neuronal por átomo, obtener la predicción atómica de dicha red neuronal y, finalmente, sumar todas las predicciones atómicas resultantes de las redes neuronales obteniendo la predicción de la configuración (caja).
2. Repetir el paso 1 hasta que no queden configuraciones a predecir.

Por lo tanto, se define una única operación:

- Época de la red neuronal: leer los datos de representación normalizados, inicializar la red neuronal y calcular la predicción.

3.6.1 - Época de la red neuronal:

Dado que se trata del mismo diseño que el del punto 3.5.1 no se repetirá el desarrollo de la época, aunque sí se debe destacar que en este caso no se calcula el error de predicción dado que sólo se

quiere obtener la predicción de la energía.

3.6.2 - Algoritmo:

Cabe señalar que este algoritmo es similar al descrito en el punto 3.5.3 con la diferencia de que, en esta ocasión, no se utiliza el adaptador sino que se llama directamente a la ejecución de la época en la GPU.

A continuación se detalla el algoritmo del módulo completo:

Algoritmo_HDNNP v:0.6; Módulo 5 → Ejecución de la red neuronal (modo predicción)
Datos de entrada → ficheros “data/MISC_DATA.dat” y “data/G_NORMALIZED_PREDICT.dat”.
Datos de salida → fichero "data/PREDICT_FINAL_OUTPUT.dat"

comienzo_algoritmo
Abrir el fichero de entrada “data/G_NORMALIZED_PREDICT.dat” en modo lectura
Leer del fichero “data/G_NORMALIZED_PREDICT.dat” el tamaño de la capa de entrada
Abrir el fichero de entrada “data/MISC_DATA.dat” en modo lectura
Leer del fichero “data/MISC_DATA.dat” el número de átomos por caja y el número de cajas de predicción
Cerrar el fichero “data/MISC_DATA.dat
Abrir el fichero de salida "data/PREDICT_FINAL_OUTPUT.dat" en modo escritura
Mientras los ficheros de entrada contengan datos
Leer NUMBER_OF_BOXES_TO_OPT configuraciones de átomos mediante sus valores de representación del fichero de entrada “data/G_NORMALIZED_PREDICT.dat”
Ejecutar la época de la red neuronal (en modo predicción)
Escribir en el fichero de salida "data/PREDICT_FINAL_OUTPUT.dat" el conjunto de energías predichas
Cerrar los ficheros de entrada
fin_algoritmo

Como se puede observar, tras la ejecución de la época de la red neuronal en la GPU, no se calcula el error de predicción sino que se escribe en el fichero de salida correspondiente las energías predichas.

3.6.2.1 - *Ejecutar la época de la red neuronal (en modo predicción), utilizando la GPU*

El algoritmo de ejecución de épocas de la red neuronal en la GPU es el mismo que se describió en el apartado 3.5.3.1 y que se denominó “Algoritmo_HDNNP v:0.5.1; Módulo 4 → ejecutar la época de la red neuronal. Versión GPU.”. Por lo tanto, y con el objeto de no describir dos veces la misma información se remite el desarrollo al punto citado anteriormente.

3.7 - Módulo 6 → Proceso de ficheros de salida:

El objetivo de este módulo es calcular el error medio de predicción y generar las gráficas que indiquen la calidad del aprendizaje y de la predicción. Por lo tanto, mediante estos datos será posible comprobar la precisión de la predicción y la calidad ajuste de la red neuronal.

3.7.1 - Algoritmo:

A continuación se detalla el algoritmo del módulo completo:

Algoritmo_HDNNP v:0.7; Módulo 6 → Proceso de ficheros de salida
    Datos de entrada → ficheros "data/ENERGY_PREDICT.dat" y "data/PREDICT_FINAL_OUTPUT.dat"
    Datos de salida → error medio de prediccion y ficheros “data/LEARNING_PROCESS.png” y “data/FINAL_PREDICTION.png”

comienzo_algoritmo
    Abrir los ficheros de entrada "data/ENERGY_PREDICT.dat" y "data/PREDICT_FINAL_OUTPUT.dat" en modo lectura
    Inicializar los acumuladores para las energías (predicha y supervisada) y los contadores de cajas (timesteps)
    Mientras los ficheros de entrada contengan datos
        Leer de "data/ENERGY_PREDICT.dat" la energía supervisada y acumularla
    Actualizar contador de caja supervisada
        Leer de "data/PREDICT_FINAL_OUTPUT.dat" la energía predicha y acumularla.
    Actualizar contador de caja predicha
    Cerrar los ficheros de entrada
    Calcular las medias de las energías supervisadas y predichas
    Imprimir a pantalla el error medio de predicción (media_energía_supervisada - media_energía_predicha)
    Ejecutar el programa externo GNUPLOT con el script “gnuplot_plot_results.in” para generar los ficheros de gráficas “data/LEARNING_PROCESS.png” y “data/FINAL_PREDICTION.png”
fin_algoritmo

CAPÍTULO 4

IMPLEMENTACIÓN

La implementación de la aplicación se ha realizado respetando el diseño descrito en el capítulo anterior y las decisiones de arquitectura y tecnología del proyecto. Por consiguiente, el programa se ha codificado en los lenguajes de programación CUDA_C/C++ y FORTRAN, utilizando la CPU para la ejecución de los procesos en serie y la GPU para la ejecución de los procesos en paralelo.

4.1 – Decisiones de implementación:

Por un lado, como decisiones de documentación del código se ha optado por generar un fichero de cabecera (.h) para cada fichero de código (.cu). En dicho fichero se encuentra una descripción del propósito (qué hace) general del código y otra para cada una de las funciones que ejecuta dicho código. Además, en cada fichero de código se encuentran comentarios que describen las operaciones que se realizan (se ha intentado repetir lo menos posible los comentarios duplicados o semejantes). Finalmente, el fichero del adaptador en FORTRAN contiene tanto la descripción general como las propias de las funciones en el mismo fichero de código (.f90)

Por otro lado, las decisiones de implementación del código son las siguientes:

1. Para cada fichero de código (.cu) se genera un fichero de cabecera (.h) con las definiciones de las funciones. De esta forma, los ficheros de código siempre importarán las cabeceras cuando necesiten utilizar las funciones de los demás ficheros de código.
2. El fichero del adaptador “wrapper.f90” no tiene fichero de cabecera.
3. Se crea un fichero de configuración (conf.h) en donde se definen todas las variables que servirán de ajuste o serán parámetros de la aplicación, como: número de las funciones de simetría, nombres de los ficheros de entrada y salida, ajustes del optimizador, tamaños de los *buffers*, etc.
4. Se genera un fichero de estructuras de datos (structs.h) en donde se definen e implementan todas las estructuras de datos necesarias de la aplicación. De esta forma, cuando un fichero de código necesite utilizar dichas estructuras sólo tendrá que importar dicho fichero de tipos de datos.
5. Se crea un sistema de mensajes de error y control con objeto de informar sobre posibles fallos en la ejecución de la aplicación. Además, este sistema permite una futura localización de la aplicación sin necesidad de cambiar código fuente ajeno a este sistema.
6. Se define un fichero *Makefile* que compila y ensambla el programa mediante el comando *make* generando todos los ficheros objeto y el binario final. También admite la opción “*clean*” para eliminar todos los ficheros objeto y el binario resultado de la compilación.
7. Se crea un fichero de entrada (gnuplot_plot_results.in) para el programa externo GNUPLOT con objeto de generar los ficheros de salida (.PNG) con las gráficas.
8. Se crea un fichero .SH con ejemplos de ejecuciones de la aplicación.
9. Los nombres de variables y de funciones siempre son lo más explícitos posibles respecto a su uso. De este modo se implementa un código lo más auto-explicativo posible evitando

comentarios innecesarios y ambigüedades.

4.2 – Ficheros de código fuente:

A continuación, se detallan todos los ficheros que componen el código fuente de la aplicación:

1. main.cu: fichero inicial de la aplicación que controla el flujo de ejecución de la misma. Es el encargado de ejecutar el algoritmo en su más alto nivel (v:0.1).
2. conf.h: fichero de configuración en donde se definen las variables de ajuste o parámetros de la aplicación tales como: número de funciones de simetría, nombres de ficheros de entrada/salida, parámetros de la red neuronal (número de neuronas, tamaño máximo de la capa de entrada, etc.), límites (tamaños de *buffers*, de las cajas, del número de cajas a procesar, número máximo de átomos por caja, etc.), ajustes del optimizador (número máximo de llamadas a la función de coste, tolerancia, etc.) y se define el carácter delimitador de los ficheros de entrada/salida (por defecto es " ", un espacio).
3. structs.h: fichero en donde se definen e implementan las estructuras de datos necesarias en determinados momentos de la aplicación. Concretamente se define una estructura para el átomo, otra para la configuración (caja) de átomos, otra para la combinación de funciones de simetría G2 y, finalmente, otra para la combinación de funciones de simetría G3.
4. messages.h/messages.cu: definen e implementan el sistema de mensajes de errores y control.
5. misc.h/misc.cu: definen e implementan aquellas funciones/operaciones que no tienen el suficiente peso como para necesitar un fichero propio. Específicamente, se desarrollan las siguientes funciones:
    1. Control de los argumentos de entrada de la aplicación: se encarga de ejecutar el algoritmo en su versión 0.2.
    2. Generación de los nombres de los ficheros de entrada respecto a los argumentos introducidos por el usuario: se encarga de ejecutar el algoritmo en su versión 0.2.
    3. Comprobación de la validez de los argumentos respecto a la expresión regular definida.
    4. Normalización de un número real (en precisión sencilla).
6. process_input_files.h/process_input_files.cu: definen e implementan la lectura de los ficheros de entrada (*ab initio*) y generan los ficheros de salida de aprendizaje ("data/LEARNING_DATA.dat") y predicción ("data/PREDICT_DATA.dat"). Se encargan de ejecutar el algoritmo en su versión 0.3.
7. load_boxes_and_generate_symmetry_functions.h/load_boxes_and_generate_symmetry_functions.cu: definen e implementan la lectura de los ficheros de entrada ("data/LEARNING_DATA.dat") y ("data/PREDICT_DATA.dat") para calcular loa valores de representación de los átomos y comenzar el pre-condicionamiento de los datos para la red neuronal. Finalmente, se generan los ficheros de salida "data/G2_SYMMETRY_LEARN.dat", "data/G3_SYMMETRY_LEARN.dat", "data/G2_SYMMETRY_PREDICT.dat", "data/G3_SYMMETRY_PREDICT.dat", "data/ENERGY_LEARN.dat" y "data/ENERGY_PREDICT.dat" [Opcional: "data/G_FORCE_RELATION_GRAPH.dat"]. Se encargan de ejecutar parte del algoritmo en su versión 0.4.
8. symmetry_functions.h/symmetry_functions.cu: definen e implementan el cálculo de las funciones de simetría para una configuración de átomos. Concretamente se trata del código

que se ejecuta en paralelo en la GPU y contendrá la llamada al *kernel*. Se encargan de ejecutar el algoritmo en su versión 0.4.1 y 0.4.2.

9. nn_preconditioning.h/nn_preconditioning.cu: definen e implementan la lectura de los ficheros de entrada “data/G2_SYMMETRY_LEARN.dat”, “data/G3_SYMMETRY_LEARN.dat”, “data/G2_SYMMETRY_PREDICT.dat”, “data/G3_SYMMETRY_PREDICT.dat”, “data/ENERGY_LEARN.dat” y “data/ENERGY_PREDICT.dat” para terminar con el pre-condicionado de los datos para la red neuronal. Finalmente se crean los ficheros de salida “data/MISC_DATA.dat”,“data/G_GRAPH.dat”, “data/G_NORMALIZED_LEARN.dat” y “data/G_NORMALIZED_PREDICT.dat”. Se encargan de ejecutar parte del algoritmo en su versión 0.4. Cabe señalar que es necesario dividir el proceso de pre-condicionado de los datos de entrada a la red neuronal (v0.4) en dos partes dado que, antes de su procesamiento, tanto la normalización como la selección de características requieren conocer todos los valores a tratar, así como sus máximos y mínimos.
10. nn_start.h/nn_start.cu: definen e implementan la lectura de los ficheros de entrada “data/MISC_DATA.dat”, “data/ENERGY_LEARN.dat”, “data/G_NORMALIZED_LEARN.dat” y “data/G_NORMALIZED_PREDICT.dat” y ejecutan la red neuronal, primero en modo aprendizaje y después en modo predicción. Finalmente se crean los ficheros de salida "data/LEARNING_PROCESS.dat" y "data/PREDICT_FINAL_OUTPUT.dat". Se encargan de ejecutar el algoritmo en sus versiones 0.5 y 0.6.
11. wrapper.f90: implementa el adaptador codificado en FORTRAN que ejecuta la red neuronal en modo aprendizaje y el optimizador de parámetros de la red neuronal (BCPOL). Se encarga de ejecutar el algoritmo en su versión 0.5.1.
12. residue.h/residue.cu: definen e implementan la ejecución de las épocas en la red neuronal tanto en modo aprendizaje como en modo predicción. Concretamente se trata del código que se ejecuta en paralelo en la GPU y contendrá la llamada al *kernel*. Se encargan de ejecutar el algoritmo en su versión 0.5.2.
13. final_results.h/final_results.cu: definen e implementan la lectura de los ficheros de entrada "data/PREDICT_FINAL_OUTPUT.dat" y “data/ENERGY_PREDICT.dat” y calculan el error de predicción medio así como generan la llamada al programa externo GNUPLOT para crear los ficheros de gráficas “data/LEARNING_PROCESS.png” y “data/FINAL_PREDICTION.png”.
14. Makefile: fichero de configuración para la llamada al comando *make*, el cual compilará y ensamblará el código fuente para generar el binario de la aplicación denominado HDNNP.
15. HDNNP.sh: *script* escrito en BASH con ejemplos de ejecuciones.
16. gnuplot_plot_results.in: *script* de entrada para el programa externo GNUPLOT que genera los ficheros de gráficas.

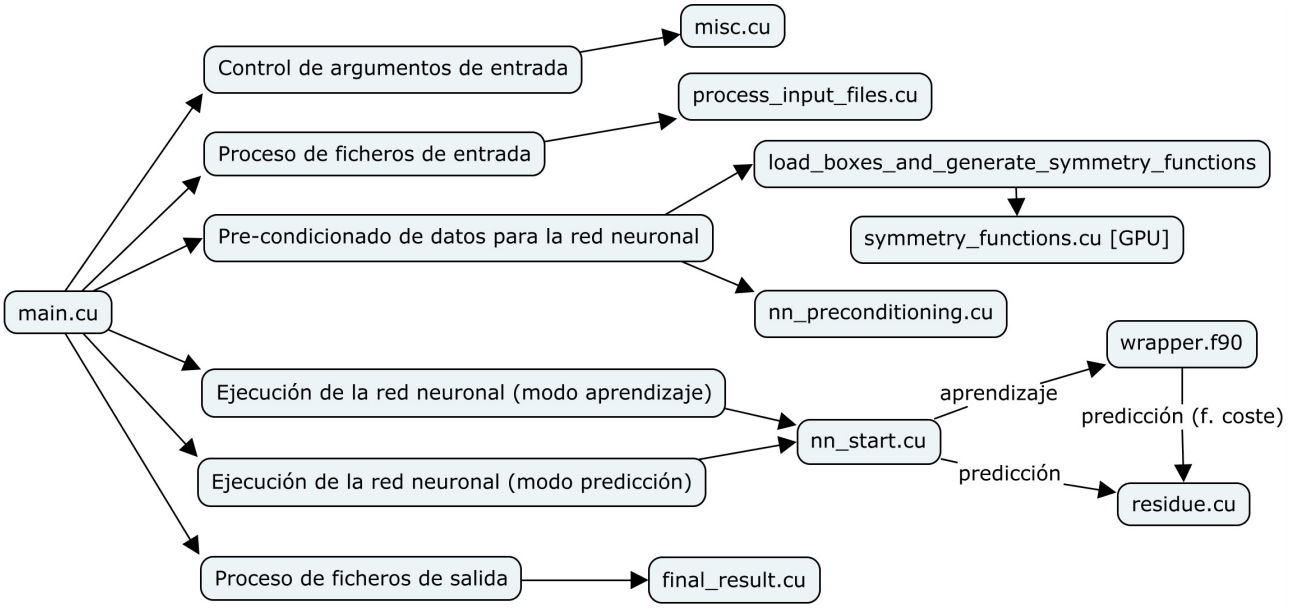


Figura 18: relación algoritmo → ficheros de código.

4.3 – Licencia y repositorios:

El código fuente de la aplicación (versión 1.0 estable) posee licencia GPLv3 (*GNU General Public License*) y se puede descargar desde los siguientes repositorios:

- https://github.com/adpozuelo/HDNNP
- https://bitbucket.org/adpozuelo/hdnnp/overview

En ambos repositorios se exponen los requisitos *hardware* y *software*, así como las instrucciones de descarga e instalación. Además, se han incluido ficheros de entrada con datos *ab initio* como ejemplo. Sin embargo, cabe señalar que sólo se han cedido 50 configuraciones por fichero dado que los ficheros originales ocupan gran cantidad de memoria física (entre uno y dos GB por fichero).

Sin embargo, en la máquina cedida por la UOC para las pruebas y simulaciones sí que se encuentran los ficheros de datos de referencia *ab initio* íntegros.

Finalmente, en la entrega final del TFG se adjunta el fichero “HDNNP_src.zip” con el código fuente en su versión 1.0 estable.

CAPÍTULO 5

PRUEBAS Y SIMULACIONES

5.1 – Pruebas:

Durante las fases de diseño e implementación de los módulos se han realizado las pruebas pertinentes con el objeto de solucionar los problemas (*bugs*) de la aplicación. Por un lado, dado el carácter modular de la aplicación no ha sido complejo aislar los problemas y solucionarlos sin que afecten al resto de la aplicación. Por otro lado, debido a que los módulos 1, 2 y 3 se pueden ejecutar una única vez y, a partir de entonces, los módulos 4, 5 y 6 son independientes de los primeros se ha podido acelerar el proceso de diseño → implementación → pruebas.

Sin embargo, cabe señalar la dificultad de depurar errores del código que se ejecuta en la GPU. Dada la naturaleza de la arquitectura y la gran cantidad de datos que se procesan en la GPU de forma paralela es extremadamente complejo localizar los errores que aparecen dentro de la GPU.

Finalmente, la aplicación superó todas las pruebas llegando a su versión estable de desarrollo 1.0.

5.2 – Simulaciones:

En primer lugar se debe resaltar que, dados los problemas afrontados durante las fases de diseño e implementación respecto con el optimizador de los parámetros de la red neuronal, las simulaciones se han realizado ajustando la red neuronal y el optimizador a los siguientes parámetros:

1. Número de valores de representación por átomo = 30.
2. Número de neuronas en cada capa oculta de la red neuronal = 4.
3. Número máximo de llamadas a la función de coste = 80000.
4. Sólo se permite predecir un único fichero de configuraciones.

El motivo de estos ajustes es reducir el tiempo necesario para realizar las simulaciones mediante la disminución de parámetros de la red neuronal y el número de llamadas a la función de coste por parte del optimizador. Por consiguiente, las operaciones más costosas computacionalmente tardan menos tiempo en ejecutarse y se pueden cumplir las fechas de entrega de los hitos correspondientes.

Dado que se dispone de seis ficheros de datos *ab initio* con las temperaturas 673K, 723K, 773K, 823K, 873K y 973K y de cinco GPUs modelos GTX590, GTX660, GTX960, GTX980 y TeslaM2075, se realizan las siguientes simulaciones:

1. En todas las GPUs como conjuntos de aprendizaje las temperaturas 673K y 973K y como conjunto de predicción la temperatura 773K. Por lo tanto, son cinco simulaciones con dos conjuntos de aprendizaje y uno de predicción.
2. En todas las GPUs como conjuntos de aprendizaje las temperaturas 673K, 723K y 973K y como conjunto de predicción la temperatura 773K. Por lo tanto, son cinco simulaciones con tres conjuntos de aprendizaje y uno de predicción.

3. En todas las GPUs como conjuntos de aprendizaje las temperaturas 673K, 723K, 873K y 973K y como conjunto de predicción la temperatura 773K. Por lo tanto, son cinco simulaciones con cuatro conjuntos de aprendizaje y uno de predicción.
4. En todas las GPUs como conjuntos de aprendizaje las temperaturas 673K, 723K, 873K, 823K y 973K y como conjunto de predicción la temperatura 773K. Por lo tanto, son cinco simulaciones con cinco conjuntos de aprendizaje y uno de predicción.

Por consiguiente, en total se realizan 20 simulaciones completas con diferentes conjuntos de aprendizaje y en diferentes modelos de GPU.

Cabe señalar que las figuras que se representarán a continuación corresponden a una única GPU elegida para representar la simulación a tratar. Sin embargo, en todas las simulaciones se expondrá una tabla con los resultados de la ejecución en todas las GPUs.

5.2.1 – Dos conjuntos de aprendizaje (673K y 973K) y uno de predicción (773K):

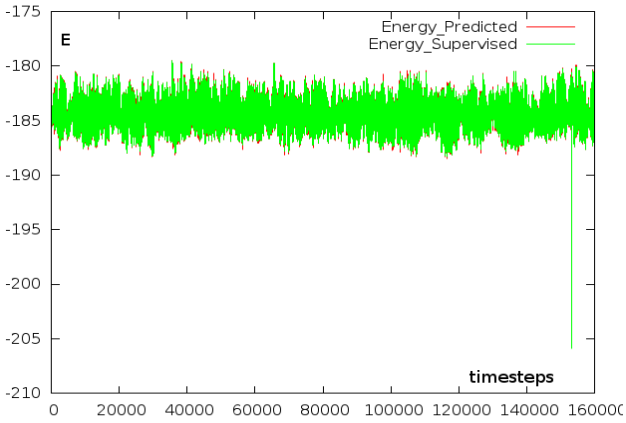


Figura 19: proceso de aprendizaje (TeslaM2075).

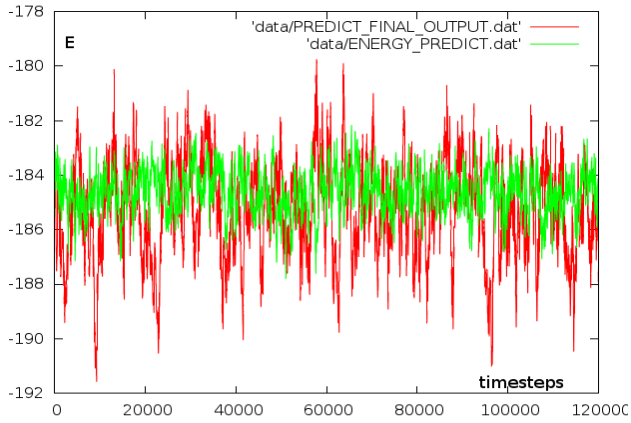


Figura 20: error de predicción (TeslaM2075).

2-Aprendizaje	Tiempo (m)	Ẽ de predicción
GTX 590	740	0.395462
GTX 660	755	0.527924
GTX 960	383	0.408463
GTX 980	340	0.553131
Tesla M2075	435	0.720734

Figura 21: tiempo de ejecución y error medio de predicción (eV) por GPU

Por un lado, como se puede observar en la figura 19 el proceso de aprendizaje es correcto dado que la diferencia entre la energía predicha (rojo), realizada tras la optimización de los parámetros de la red neuronal, y la energía supervisada (verde) converge y es pequeña.

Por otro lado, en la figura 20 se representa la predicción de la energía del fichero de configuraciones a predecir (rojo) una vez que la red neuronal ha aprendido de todos los conjuntos de entrenamiento. Tal y como se puede observar el ajuste no es óptimo dado que la predicción no se asemeja a la energía supervisada (verde).

Finalmente, en la figura 21 se exponen todas las simulaciones, para este caso concreto, en las diferentes GPUs. Concretamente se puede comprobar que el error medio de predicción ($\bar{E}$) es alto lo que explica el mal ajuste representado en la figura 20.

Por lo tanto, observando las gráficas y la tabla de resultados podemos concluir que el sistema aprende correctamente pero no predice con el ajuste deseado. Además, se puede comprobar que las GPUs más modernas y/o potentes necesitan menos tiempo para realizar la simulación y que todas ellas poseen un error medio de predicción similar (del mismo orden de magnitud).

5.2.2 – Tres conjuntos de aprendizaje (673K, 723K y 973K) y uno de predicción (773K):

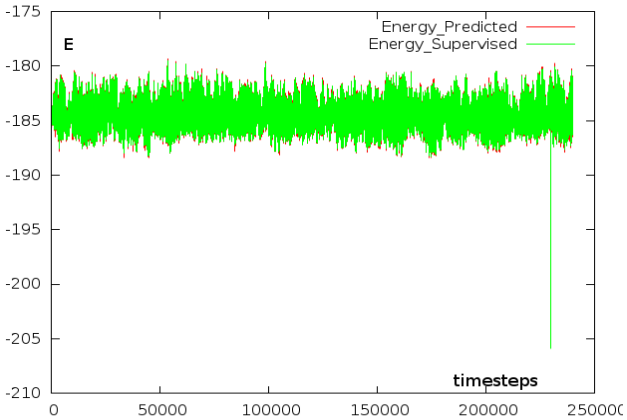


Figura 22: proceso de aprendizaje (GTX590).

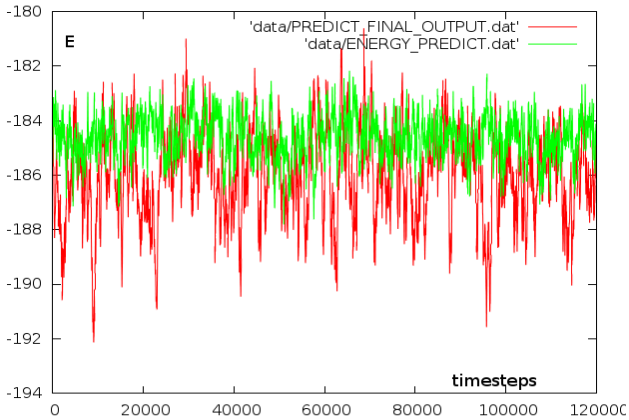


Figura 23: error de predicción (GTX590).

3-Aprendizaje	Tiempo (m)	$\bar{E}$ de predicción
GTX 590	1087	1.399277
GTX 660	1034	1.256638
GTX 960	488	1.067123
GTX 980	307	1.328995
Tesla M2075	576	0.819427

Figura 24: tiempo de ejecución y error medio de predicción (eV) por GPU

Por un lado, como se puede observar en la figura 22 el proceso de aprendizaje es correcto dado que la diferencia entre la energía predicha (rojo), realizada tras la optimización de los parámetros de la red neuronal, y la energía supervisada (verde) converge y es pequeña.

Por otro lado, en la figura 23 se representa la predicción de la energía del fichero de configuraciones a predecir (rojo) una vez que la red neuronal ha aprendido de todos los conjuntos de entrenamiento. Tal y como se puede observar el ajuste no es óptimo dado que la predicción no se asemeja a la energía supervisada (verde).

Finalmente, en la figura 24 se exponen todas las simulaciones, para este caso concreto, en las diferentes GPUs. Concretamente se puede comprobar que el error medio de predicción ($\bar{E}$) es alto (incluso más que en el caso anterior) lo que explica el mal ajuste representado en la figura 23.

Por lo tanto, observando las gráficas y la tabla de resultados podemos concluir que el sistema aprende correctamente pero no predice con el ajuste deseado. Además, se puede comprobar que las GPUs más modernas y/o potentes necesitan menos tiempo para realizar la simulación y que todas ellas (excepto la Tesla) poseen un error medio de predicción similar (del mismo orden de magnitud).

5.2.3 – Cuatro conjuntos de aprendizaje (673K, 723K, 873K y 973K) y uno de predicción (773K):

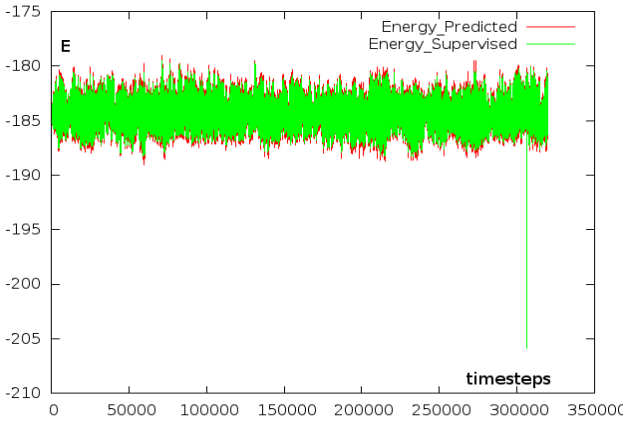


Figura 25: proceso de aprendizaje (GTX660).

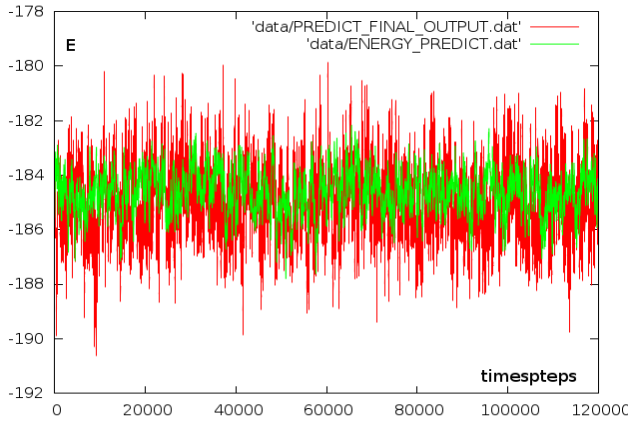


Figura 26: error de predicción (GTX660).

4-Aprendizaje	Tiempo (m)	Ē de predicción
GTX 590	1322	0.320465
GTX 660	1399	0.396881
GTX 960	763	0.638229
GTX 980	409	0.631714
Tesla M2075	732	0.707733

Figura 27: tiempo de ejecución y error medio de predicción (eV) por GPU

Por un lado, como se puede observar en la figura 25 el proceso de aprendizaje es correcto dado que la diferencia entre la energía predicha (rojo), realizada tras la optimización de los parámetros de la red neuronal, y la energía supervisada (verde) converge y es pequeña.

Por otro lado, en la figura 26 se representa la predicción de la energía del fichero de configuraciones a predecir (rojo) una vez que la red neuronal ha aprendido de todos los conjuntos de entrenamiento. Tal y como se puede observar el ajuste no es óptimo (aunque es mejor que en los casos anteriores) dado que la predicción no se asemeja a la energía supervisada (verde).

Finalmente, en la figura 27 se exponen todas las simulaciones, para este caso concreto, en las diferentes GPUs. Concretamente se puede comprobar que el error medio de predicción (Ē) es alto (semejante al primer caso) lo que explica el mal ajuste representado en la figura 26.

Por lo tanto, observando las gráficas y la tabla de resultados podemos concluir que el sistema aprende correctamente pero no predice con el ajuste deseado. Además, se puede comprobar que las

GPUs más modernas y/o potentes necesitan menos tiempo para realizar la simulación y que todas ellas poseen un error medio de predicción similar (del mismo orden de magnitud).

5.2.4 – Cinco conjuntos de aprendizaje (673K, 723K, 823K, 873K y 973K) y uno de predicción (773K):

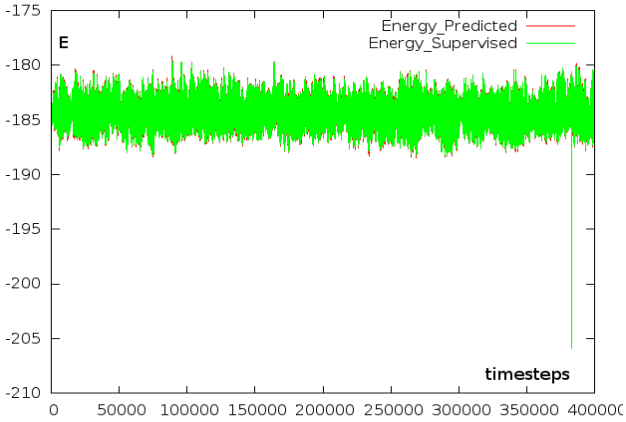


Figura 28: proceso de aprendizaje (GTX960).

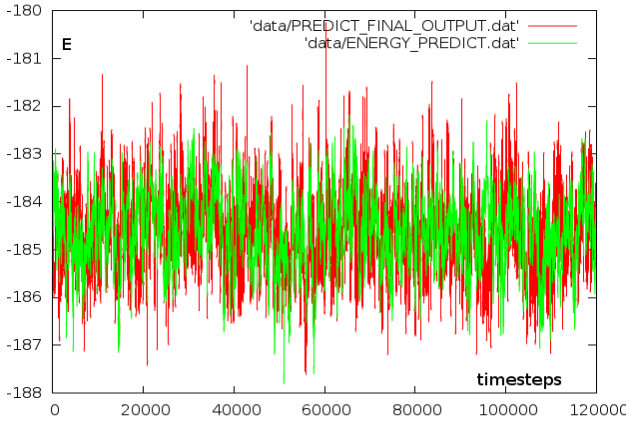


Figura 29: error de predicción (GTX960).

5-Aprendizaje	Tiempo (m)	Ẽ de predicción
GTX 590	1371	0.340820
GTX 660	1726	0.219604
GTX 960	935	0.007904
GTX 980	603	0.101883
Tesla M2075	1019	0.048141

Figura 30: tiempo de ejecución y error medio de predicción (eV) por GPU

Por un lado, como se puede observar en la figura 28 el proceso de aprendizaje es correcto dado que la diferencia entre la energía predicha (rojo), realizada tras la optimización de los parámetros de la red neuronal, y la energía supervisada (verde) converge y es pequeña.

Por otro lado, en la figura 29 se representa la predicción de la energía del fichero de configuraciones a predecir (rojo) una vez que la red neuronal ha aprendido de todos los conjuntos de entrenamiento. Tal y como se puede observar el ajuste es óptimo dado que la predicción se asemeja a la energía supervisada (verde).

Finalmente, en la figura 30 se exponen todas las simulaciones, para este caso concreto, en las diferentes GPUs. Concretamente se puede comprobar que el error medio de predicción (Ẽ) es más bajo que en los casos anteriores, siendo realmente óptimo en los casos de la GTX960 (0.007904 eV) y de la Tesla (0.048141 eV).

Por lo tanto, observando las gráficas y la tabla de resultados podemos concluir que el sistema aprende correctamente y predice con el ajuste deseado (concretamente en dos casos). Además, se puede comprobar que las GPUs más modernas y/o potentes necesitan menos tiempo para realizar la simulación y que el error medio de predicción no tiene el mismo orden de magnitud.

5.3 – Rendimiento:

Para evaluar el rendimiento de la aplicación se ha utilizado la herramienta NVIDIA Visual Profiler incluida en el SDK de CUDA. A continuación se exponen los resultados:

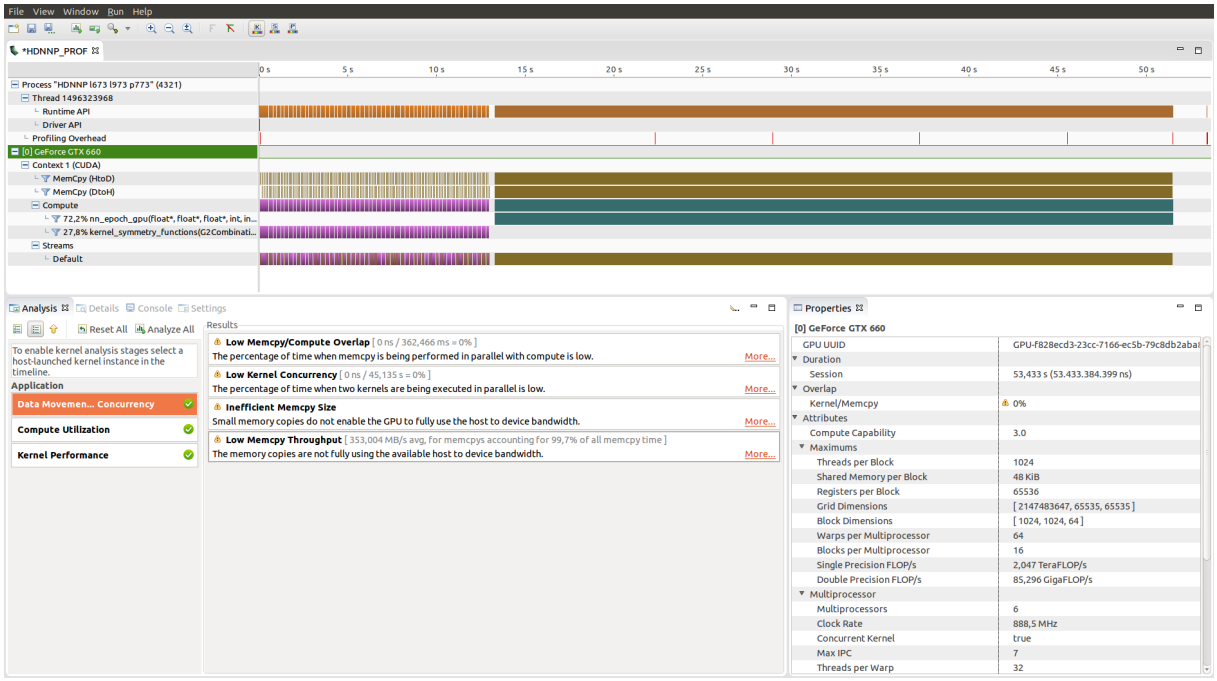


Figura 31: *timeline* de la aplicación y concurrencia de datos.

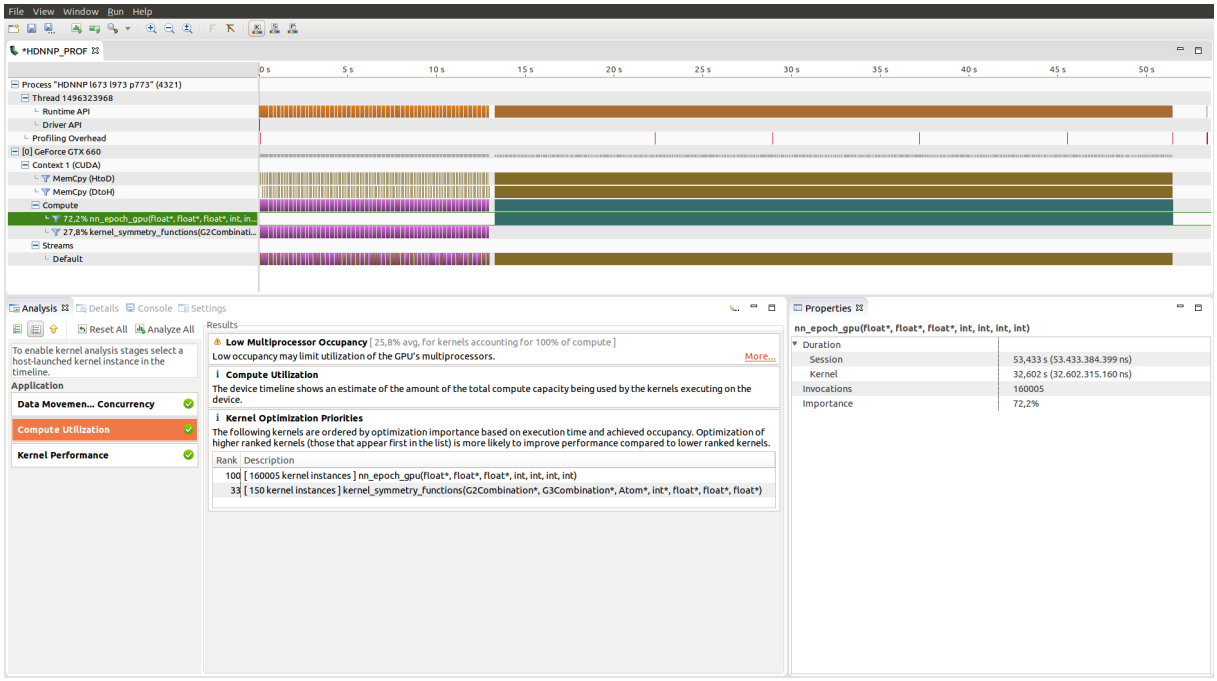


Figura 32: *timeline* de la aplicación y utilización computacional.

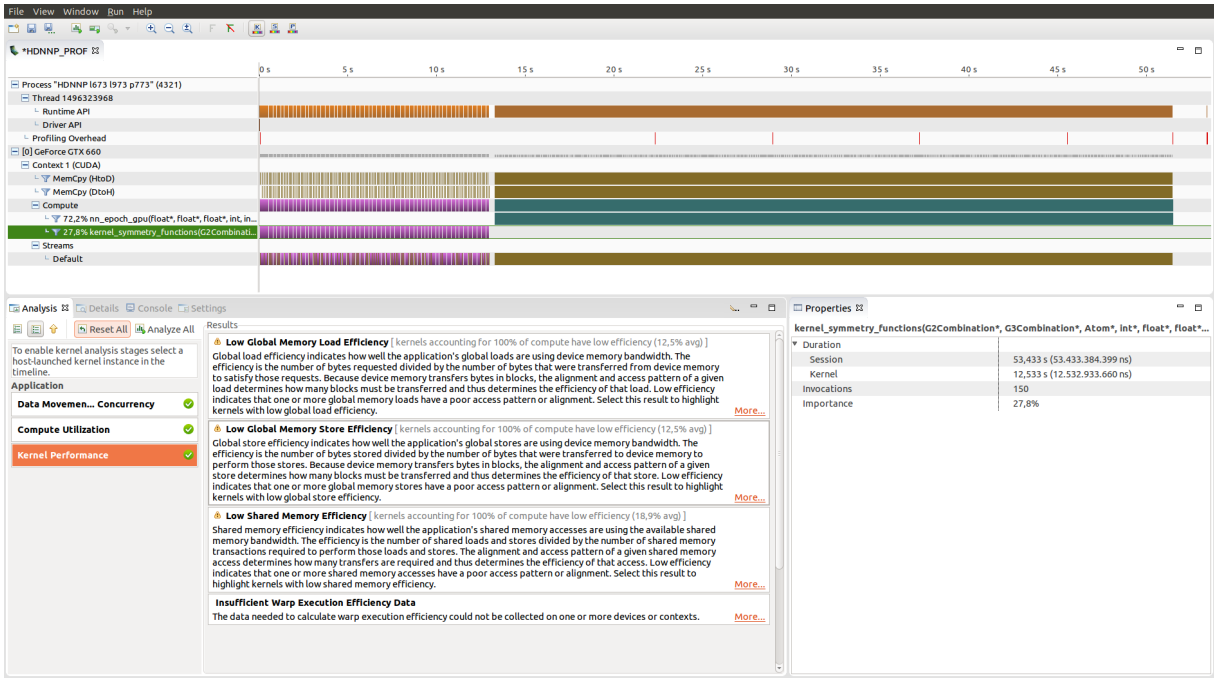


Figura 33: *timeline* de la aplicación y rendimiento de los *kernels*.

Por un lado, señalar que la simulación para la evaluación del rendimiento se realizó con dos conjuntos de aprendizaje (50 configuraciones de átomos en cada fichero) y con un conjunto de predicción (50 configuraciones de átomos) dado el alto volumen de datos que genera la aplicación NVIDIA Visual Profiler.

Por otro lado, tal y como se puede observar en las figuras superiores, la aplicación supera el *test* de rendimiento (*sticks* verdes en todas las fases) aunque se obtienen algunas recomendaciones. Dichas recomendaciones tienen relación con el diseño de la aplicación, concretamente:

1. En las figura 31, 32 y 33 se indica que:
    1. Hay poca eficiencia en la copia de datos entre CPU → GPU y viceversa: esto es debido a que las simulaciones se han realizado con "sólo" 64 átomos. Según se aumente el número de átomos por configuración aumentará el tamaño de los datos a copiar y tanto la eficiencia como el *throughput* aumentará.
    2. Se han utilizado pocos multi-procesadores: es el mismo caso anterior excepto que es debido al número de cajas que se procesan concurrentemente.
    3. La eficacia en el uso de la memoria (global y compartida) es poca: este mensaje procede de la suma de los dos anteriores. Según se aumente el número de átomos por configuración y el número de configuraciones a procesar paralelamente, aumentará la eficacia en el uso de las memorias.
    4. No se utiliza el *overlap* en la copia de datos ni la concurrencia de *kernels*: son opciones que no se han utilizado dado que dependen del *hardware* y se ha intentado realizar una implementación lo más compatible posible con todas las GPUs.

Finalmente, en los repositorios de código (Github y Bitbucket) se adjunta el fichero

"HDNNP_PROFILING.xml" mediante el cual se puede reproducir esta evaluación con la herramienta NVIDIA Visual Profiler.

CAPÍTULO 6

CONCLUSIONES

6.1 – Error medio de predicción:

Por un lado, se debe analizar el error medio de predicción obtenido de las simulaciones observando la siguiente gráfica:

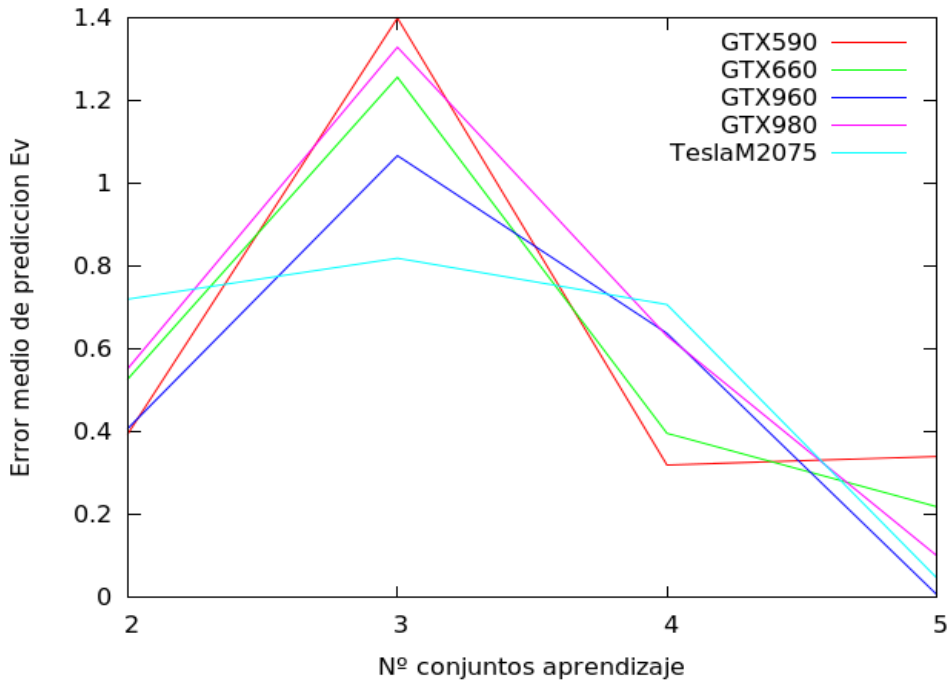


Figura 34: Error medio de predicción Ē (eV) por cada GPU.

En la figura 34 podemos observar como el error medio de predicción disminuye según se incrementan los conjuntos de aprendizaje con los que se alimenta la red neuronal. Concretamente, con cinco conjuntos de aprendizaje ya se obtiene un error medio de predicción óptimo.

Cabe señalar que los distintos errores en cada tipo de GPU pueden estar relacionados con la diferencia en el número de núcleos de cada *hardware*. En el caso de la GPU Tesla, se deben tener presentes las modificaciones específicas que NVIDIA ha implementado en este tipo de *hardware* destinado al cálculo científico.

6.2 – Tiempo:

Por otro lado, se debe analizar el tiempo necesario para realizar las simulaciones mediante la siguiente gráfica:

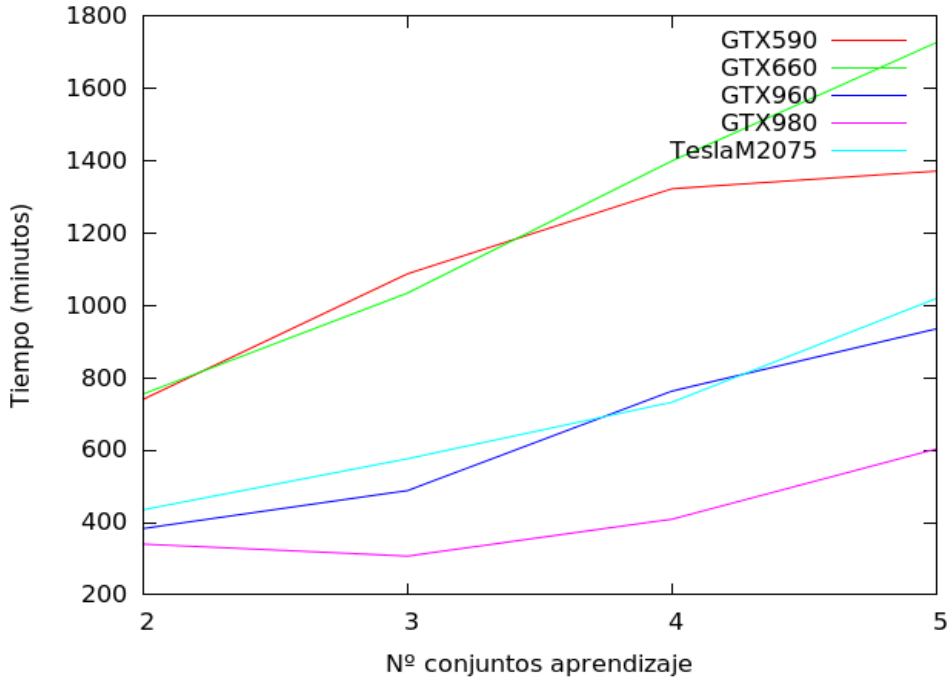


Figura 35: tiempo necesitado para realizar la simulación por cada GPU.

En la figura 35 podemos comprobar como el tiempo necesario para realizar la simulación aumenta de forma más o menos lineal respecto al número de conjuntos de aprendizaje utilizados.

6.3 – Conclusiones finales:

Finalmente, se concluye que:

1. La red neuronal aprende correctamente en todas las situaciones.
2. El tiempo de simulación aumenta de forma, más o menos, lineal respecto del número de conjuntos de aprendizaje utilizados para alimentar a la red neuronal.
3. La predicción mejora cuantos más conjuntos aprende la red neuronal, llegando a ser óptima con cinco conjuntos.

Por lo tanto, como cualquier sistema inteligente la red neuronal HDNNP necesita tener más experiencia (ver más cosas) para tomar mejores decisiones (realizar predicciones).

6.4 – El futuro:

Desde el punto de vista del ajuste de la red neuronal se realizarán simulaciones variando los parámetros de la misma (tamaño de la capa de entrada, neuronas, etc.) con objeto de comprobar si con más parámetros entre neuronas y BIAS se consigue un mejor ajuste con menos conjuntos de aprendizaje. Sin embargo, estas simulaciones requieren mucho tiempo dado que según se aumentan los parámetros de la red neuronal el optimizador necesita mucho más tiempo para ajustarlos. Por lo tanto, y debido a que no se tiene dicho tiempo, no se han podido realizar para este proyecto.

Desde el punto de vista del diseño, y concretamente del optimizador, se realizarán dos acciones:

1. Se utilizará una nueva función de coste en el proceso de optimización que, junto a la diferencia de energías predichas y supervisadas, incorpore la diferencia entre los gradientes de dichas energías como estimación de las fuerzas sobre cada átomo y las fuerzas *ab initio* de las que se dispone en los datos de referencia.
2. Se estudiará, diseñará e implementará un filtro extendido de Kalman con objeto de sustituir el actual optimizador.

Desde el punto de vista de los datos se realizarán simulaciones variando el número de átomos por configuración, con objeto de comprobar el ajuste de la red neuronal de alta dimensionalidad con conjuntos de átomos de distinto tamaño. Para realizar esta tarea se solicitarán más datos *ab initio* al Dr. Nebil A. Katcho.

Por lo tanto, y dado el carácter práctico de este proyecto se seguirá trabajando en él dedicándole el tiempo que se pueda.

ANEXO

A.1 - Datos de entrada → temperaturas de los sistemas para aprender y a predecir:

Los datos de entrada corresponden a los argumentos introducidos en la ejecución del programa.

Por un lado, se tienen las temperaturas (en grados Kelvin) de los sistemas de átomos con las que la red neuronal debe aprender y, por el otro lado, se tienen las temperaturas (en grados Kelvin) de los sistemas de átomos a predecir. Ambos datos de entrada deben tener la misma forma, que se especifica de la siguiente manera:

- En primer lugar, el dato de entrada debe tener una “l” (*learning*) o una “p” (*predict*).
- En segundo lugar, el dato de entrada debe tener la temperatura a aprender/predecir con un mínimo de un carácter y un máximo de cuatro [0 , 9999].
- Cualquier otro dato de entrada debe ser rechazado.

Por lo tanto, las reglas anteriores se describen mediante la siguiente expresión regular:

“^(l|p)[0-9]{1,4}$”

Finalmente, se detallan los límites en cuanto al número de datos:

- Número mínimo de datos de entrada: un dato para aprender y un dato para predecir.
- Número máximo de datos de entrada: sin límite.

A.2 - Datos de salida → energías de los sistemas a predecir:

Los datos de salida corresponden a las energías de los sistemas de átomos a predecir.

Para cada dato de entrada (temperatura) del tipo predicción se generará un fichero de salida, en el directorio “*data*” de la aplicación, que contendrá una línea con el valor de la energía del conjunto de átomos dado. El nombre del fichero de salida se especifica según la siguiente forma:

- En primer lugar, el nombre empezará con “E.Te.”.
- En segundo lugar, el nombre tendrá la temperatura a predecir.
- Finalmente, el nombre del fichero terminará con una “K”.

El nombre de dichos ficheros se generará en el módulo de control de argumentos de entrada.

A.3 – Cálculo de otras propiedades macroscópicas:

Una vez obtenida la energía de una configuración atómica se pueden estimar las demás propiedades macroscópicas utilizando dicha energía. A continuación se expone, como ejemplo, el cálculo de la presión debida a los átomos del sistema a simular.

A partir de $E_{tot} = E[\{G_i\}]$ (energía expresada en términos de los valores de representación obtenidos

de evaluar las funciones de simetría) la presión viene dada por:

$$\Pi_{\alpha\beta} = -\frac{1}{V}\sum_{\lambda}\frac{\delta E_{tot}}{\delta H_{\alpha\lambda}}[H^T]_{\lambda\beta}$$ donde,

- V es el volumen de la caja de simulación.
- α, β y λ corresponden a x, y, z.
- H es un matriz que tiene como elementos $(\vec{a}_1, \vec{a}_2, \vec{a}_3)$, siendo $\vec{a}_i$ los tres vectores que determinan la geometría de la caja de simulación. El super-índice T designa la matriz traspuesta.

Concretamente, para el caso el Teluro estudiado en este proyecto la caja de simulación es cúbica por lo que:

$$H^T = \begin{pmatrix} l & 0 & 0 \\ 0 & l & 0 \\ 0 & 0 & l \end{pmatrix}$$ siendo l el lado de la caja.

Por lo tanto, siendo $\Pi_{\alpha\beta}$ los elementos del tensor de presiones, la magnitud a buscar es:

$$P = \frac{1}{3}(\Pi_{xx} + \Pi_{yy} + \Pi_{zz})$$

Finalmente, cabe señalar que las derivadas de la expresión anterior sobre la red neuronal son complejas, aunque como comprobación se pueden evaluar numéricamente mediante un método de diferencias finitas.

AGRADECIMIENTOS

- A mi familia por su paciencia y apoyo.
- Al Profesor de Investigación D. Enrique Lomba García por su apoyo, ayuda y orientación.
- Al Dr. Nebil A. Katcho por la cesión de los datos de referencia *ab initio*.
- Al departamento de Mecánica Estadística y Materia Condensada del Instituto de Química Física Rocasolano (CSIC) por permitirme usar el sistema HPC (*High-Performance Computing*) Ladon-Hidra para realizar las simulaciones.
- Al departamento de Bio-informática Estructural del Instituto de Química Física Rocasolano (CSIC) por su apoyo y orientación.

BIBLIOGRAFÍA

1. "Constructing High-Dimensional Neural Network Potentials: A Tutorial Review", Jörg Behler. *Int. J. Quantum Chem*. **2015**, *115*, 1032-1050. DOI: 10.1002/qua.24890
2. "Neural Network Models of Potential Energy Surfaces: Prototypical Examples", James B. Witkoskie and Douglas J. Doren. *Journal of Chemical Theory and Computation.* **2005**, *1 (1)*, 14-23. DOI: 10.1021/ct049976i
3. "Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces", Behler y Parrinello, Phys. Rev. Lett. **2007**, *98*, 146401; DOI: 10.1103/PhysRevLett.98.146401
4. "Neural network models of potential energy surfaces", Thomas B. Blank, Steven D. Brown, August W. Calhoun, and Douglas J. Doren. J. Chem. Phys. **1995,** *103*, 4129; DOI: 10.1063/1.469597.
5. "Atom-centered symmetry functions for constructing high-dimensional neural network potentials", Jörg Behler. J. Chem. Phys. **2011**, *134*, 074106; DOI: 10.1063/1.3553717.
6. CUDA Programming - ISBN: 978-0-12-415933-4.
7. CUDA By Example - ISBN 978-0-13-138768-3.
8. CUDA Toolkit Documentation – http://docs.nvidia.com/cuda/index.html.