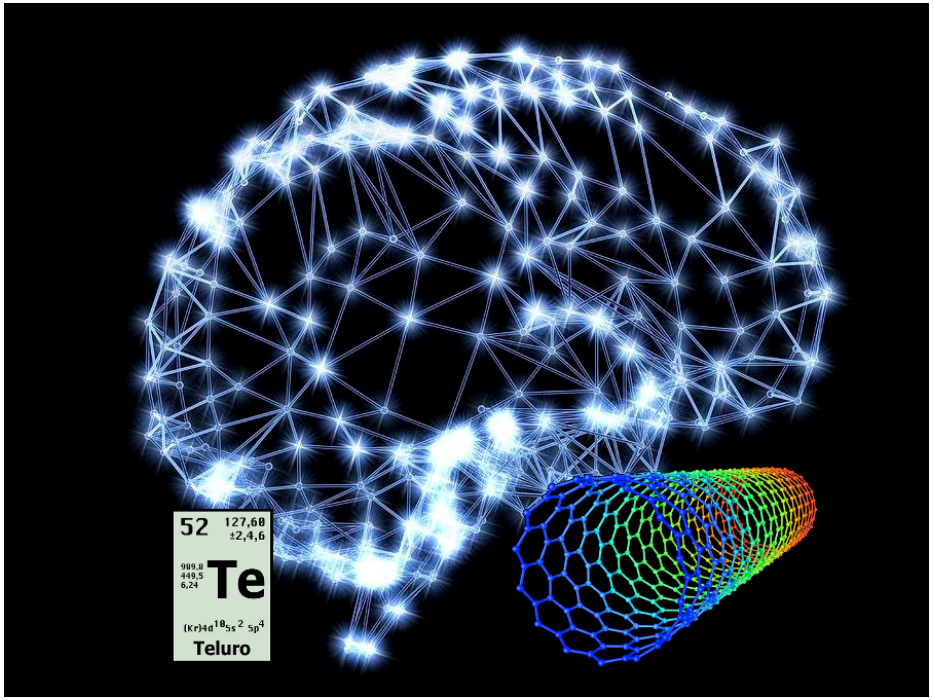



Antonio Díaz Pozuelo

adpozuelo@uoc.edu

TFG – Grado en Ingeniería Informática

Universitat Oberta de Catalunya

Enero de 2016

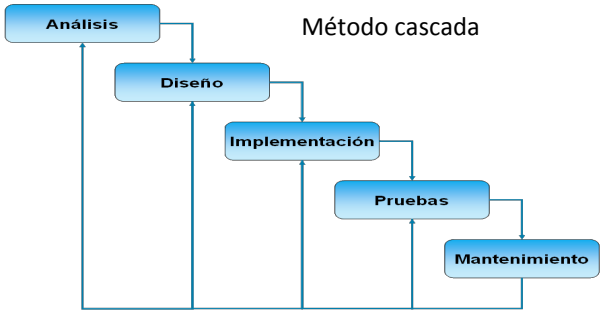
# High-Dimensional Neural Network Potentials

# 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 NVIDA 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.

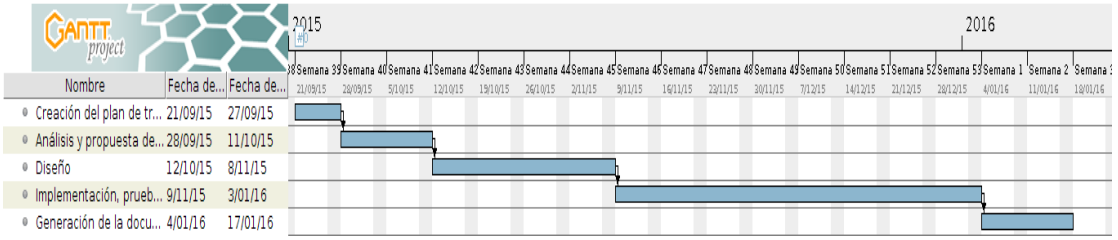
# PLANIFICACIÓN

- 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.




Método cascada

Diagrama de Gantt



# ANÁLISIS Y PROPUESTA DE SOLUCIONES

- Exposición del problema:
  - Dado un sistema de átomos (configuración) se quieren calcular las propiedades macroscópicas (energía, presión, conductividad, etc.) del mismo.
- Posibles soluciones:
  - 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).
  - 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.
  - 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.

# ANÁLISIS Y PROPUESTA DE SOLUCIONES

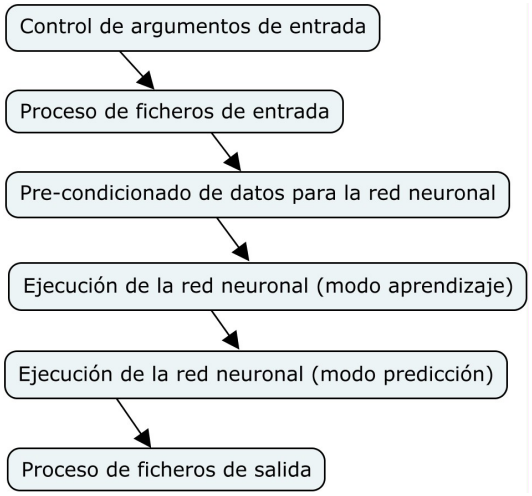
- Solución elegida:
  - Para resolver el problema dado se utilizará la solución ofrecida por las redes neuronales debido a que:
    - Se desea aprender sobre el diseño y desarrollo de redes neuronales.
    - Es el campo menos desarrollado de los tres citados anteriormente.
    - Existe la necesidad en 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:
  - 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.
  - La red neuronal será multi-capa (contendrá varias capas ocultas) con el objeto de obtener una mayor precisión, flexibilidad y capacidad de ajuste.
  - 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.

# 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:
  - Se desea aprender el desarrollo de aplicaciones utilizando esta tecnología.
  - La paralelización en GPU es una tecnología en continua expansión en el ámbito científico.
  - 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:
  - Se desea aprender a implementar aplicaciones en CUDA_C/C++.
  - Se trata de GPUs de ámbito general que se pueden encontrar en gran parte del ecosistema informático existente.
  - Disponen de un SDK (*Software Development Kit*) maduro y basado en el lenguaje de programación C.

# 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.
- Algoritmo general:



# 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.
- 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.
- 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.

# Módulo 2 → Proceso de ficheros de entrada

```
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
```

HISTORY.Te.673K.

```
  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
```

E.Te.673K

- 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 / $ps^2$).
- 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.

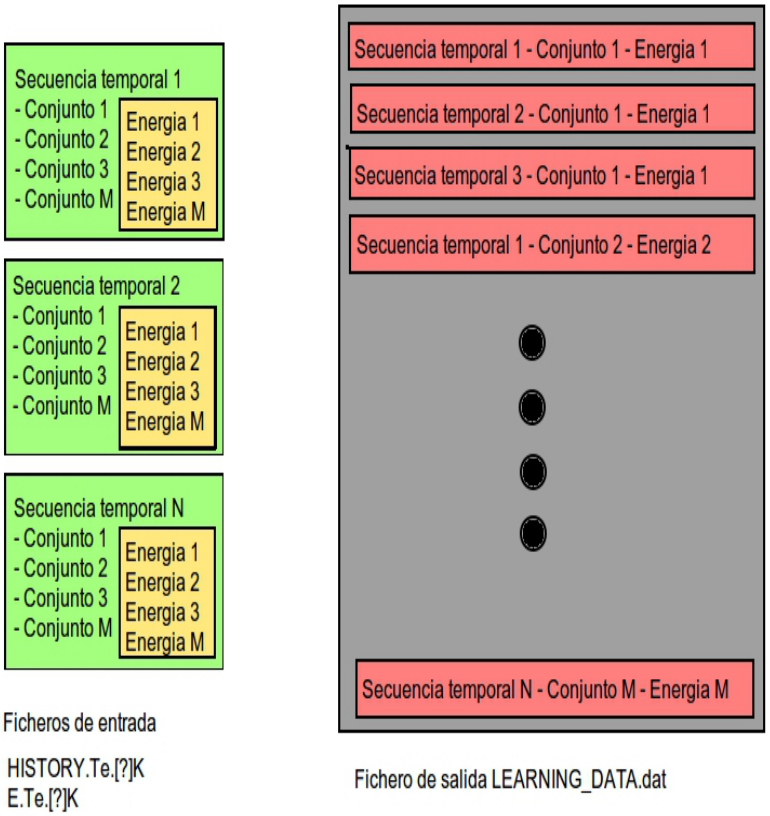
# Módulo 2 → Proceso de ficheros de entrada

- 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:
  - 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.
  - Dimensiones de la caja: las condiciones de contorno incluyen periodicidad, por lo que cada caja de átomos es periódica en el espacio.
  - Coordenadas del átomo: describen la posición de cada átomo en el espacio (x, y, z) respecto a un vértice de la caja de simulación.
  - 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:
  - 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.
- 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).

```
-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
```

data/LEARNING_DATA.dat

# Módulo 2 → Proceso de ficheros de entrada



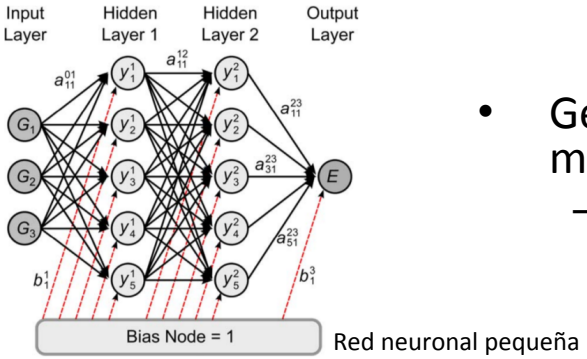

Estructura del fichero de salida del aprendizaje

- 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.

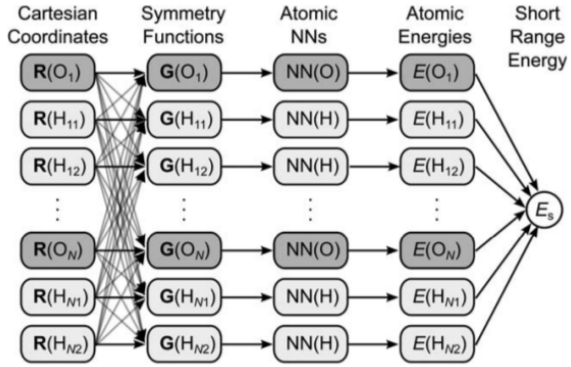
# Módulo 3 → Pre-condicionado de datos para la red neuronal

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

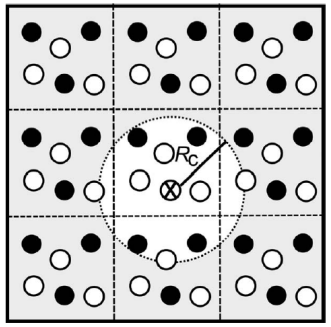
# Módulo 3 → Pre-condicionado de datos para la red neuronal




Red neuronal pequeña




Red neuronal alta dimensionalidad




Cutoff del átomo

- Generar los valores de representación de cada átomo mediante funciones de simetría:
  - 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 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 el vector de entradas $\{G_i\}$ corresponde al conjunto de <u>valores de representación</u>, 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". Dicho *cutoff* define el entorno energéticamente relevante del átomo y se establece como el radio de la esfera que limita el entorno.

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

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

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

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

  - ✓ Función radial: esta función define el ordeń 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.

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

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

  - ✓ 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.

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

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.
$\theta_{ijk}$: ángulo, respecto al átomo central i, que forman los tres átomos i, j y k.

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

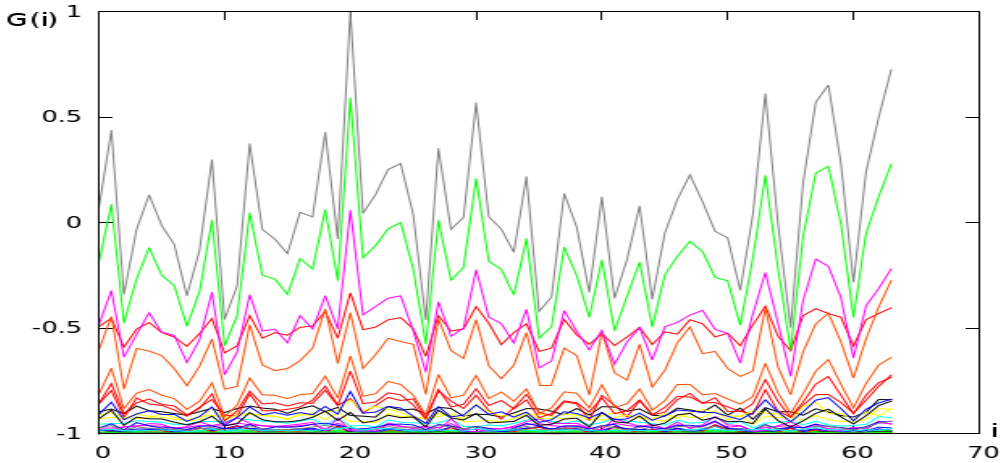
- Finalmente, la operación de cálculo de funciones de simetría consistirá en evaluar cada unas de las funciones anteriores para cada átomo de la configuración y para cada combinación de parámetros posible. De este modo, cada átomo quedará representado por el conjunto de valores obtenidos del cálculo 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.
- Para los parámetros de ajuste mencionados anteriormente, se usarán los siguientes valores (son los definidos en el artículo de Behler[1]):
  - $\eta = \{ 0, 0.04, 0.14, 0.32, 0.71, 1.79 \}$
  - $R_s = \{ 0, 1, 2, 3, 4, 5 \}$
  - $\zeta = \{ 1, 2, 4, 16 \}$
  - $\lambda = \{ -1, 1\}$
- 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.

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

- 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 $d_{ij}$, entonces si $d_{ij} \rightarrow 0$, $f_{ij} \rightarrow 0$. 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

$$d_{ij} = \sqrt{\sum_{\alpha} \left(G_i^2 - G_j^2\right)^2}$$

$$F_{ij} = \left|\vec{F}_i - \vec{F}_j\right|$$




valores de representación normalizados

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

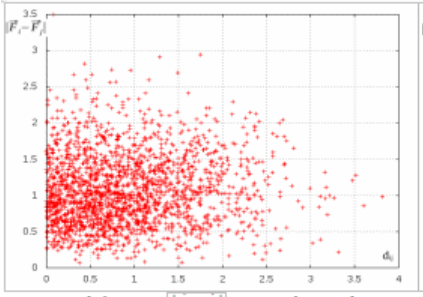


Figura 10: $d_{ij}$ frente a $|\vec{F}_i - \vec{F}_j|$ con un único valor.

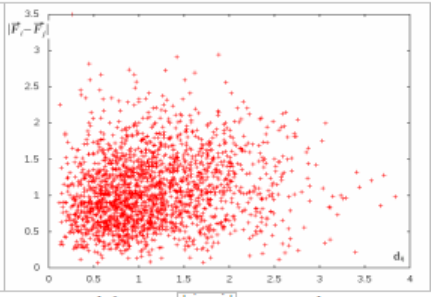


Figura 11: $d_{ij}$ frente a $|\vec{F}_i - \vec{F}_j|$ con 15 valores.

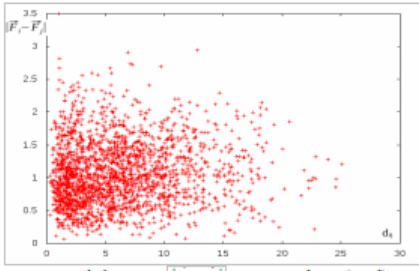


Figura 12: $d_{ij}$ frente a $|\vec{F}_i - \vec{F}_j|$ con 30 valores (total).

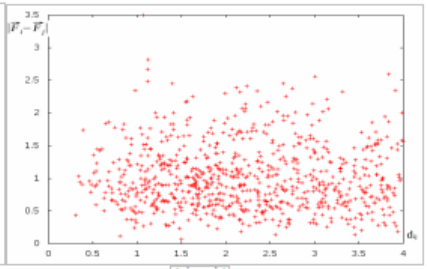


Figura 13: $d_{ij}$ frente a $|\vec{F}_i - \vec{F}_j|$ con 30 valores (detalle).

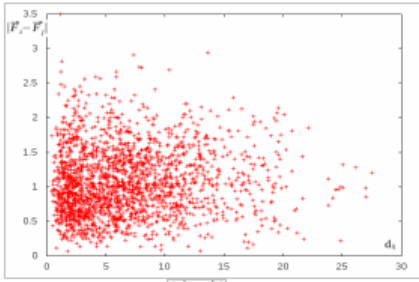


Figura 14: $d_{ij}$ frente a $|\vec{F}_i - \vec{F}_j|$ con 57 valores (total).

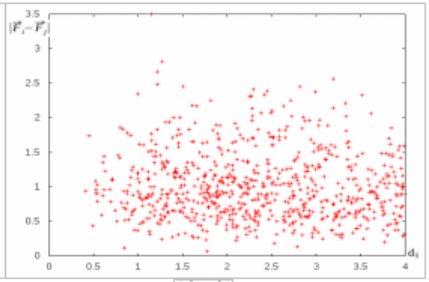


Figura 15: $d_{ij}$ frente a $|\vec{F}_i - \vec{F}_j|$ con 57 valores (detalle).

- Si se utiliza un único valor de representación (figura 10), 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 $f_{ij} \neq 0$ , 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.

# Módulo 3 → Pre-condicionado de datos 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):

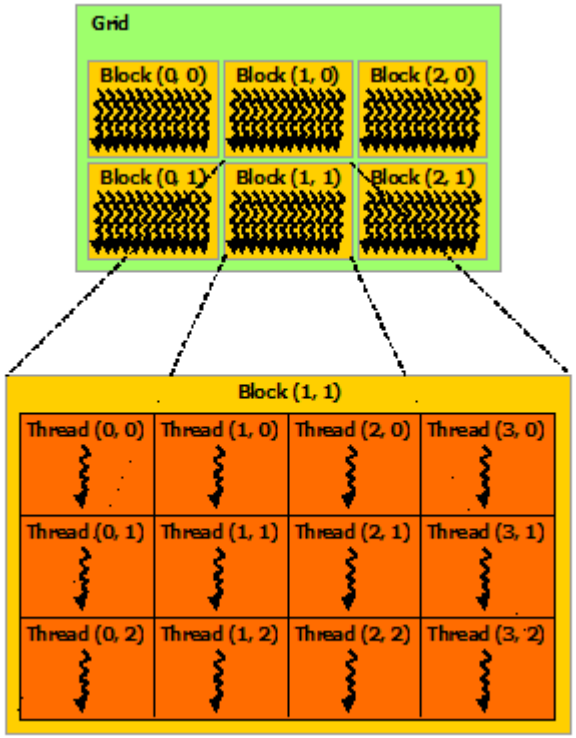
$$G_i^{\text{scaled}} = \frac{2\left(G_i - G_{i,\min}\right)}{G_{i,\max} - G_{i,\min}} - 1,$$

- 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).

$$y = \frac{1}{1 + e^{-x}}$$ Función sigmoide

- Finalmente, se escribirán los ficheros de salida correspondientes.

# Módulo 3 → Paralelización GPU




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);

# Módulo 3 → Paralelización GPU

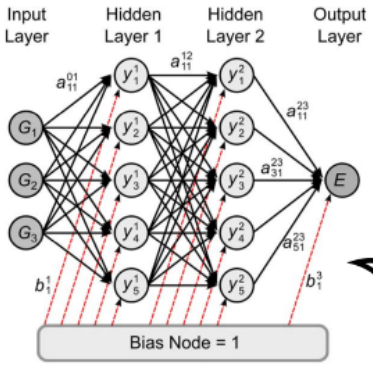
- Reseñas:
  - Se utiliza un acumulador en memoria compartida con el objetivo de maximizar el uso de los registros de la GPU y minimizar los acceso a la memoria global de la GPU. Mediante esta técnica se optimiza el acceso a memoria aunque los hilos tengan que cooperar entre ellos. Por lo tanto, sólo se accede a la memoria global en el momento de almacenar los acumuladores de cada átomo para cada combinación de las funciones de simetría.
  - El cálculo de distancias entre átomos se debe realizar teniendo en cuenta la periodicidad de la caja.
  - En la evaluación de la función de simetría G3, el cálculo del coseno entre los tres átomos se realiza mediante la regla del coseno dado que se tienen las distancias entre átomos.
  - 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.
  - 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 G3 (regla del coseno).
- 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.

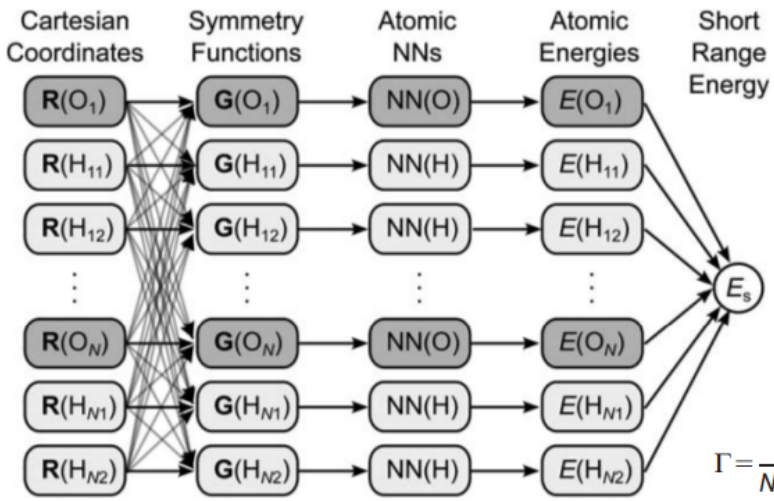
# Módulo 4 → Ejecución de la red neuronal (modo aprendizaje):




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

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

Época de la red neuronal




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

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

- 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 $E_s$ (7). Este valor será utilizado para calcular el error de la predicción (8) donde 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).

- 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|). 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 dado que se necesita una función no lineal. 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.

# Módulo 4 → Ejecución de la red neuronal (modo 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.
- 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).
- 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);

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

- Reseñas:
  - 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.
  - 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.
  - 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).
  - 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.
  - 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).
  - 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.
- 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.

# 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. Dado que se trata del mismo diseño que el del punto anterior (época en modo aprendizaje) 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.

# Módulo 6 → Proceso de ficheros de salida

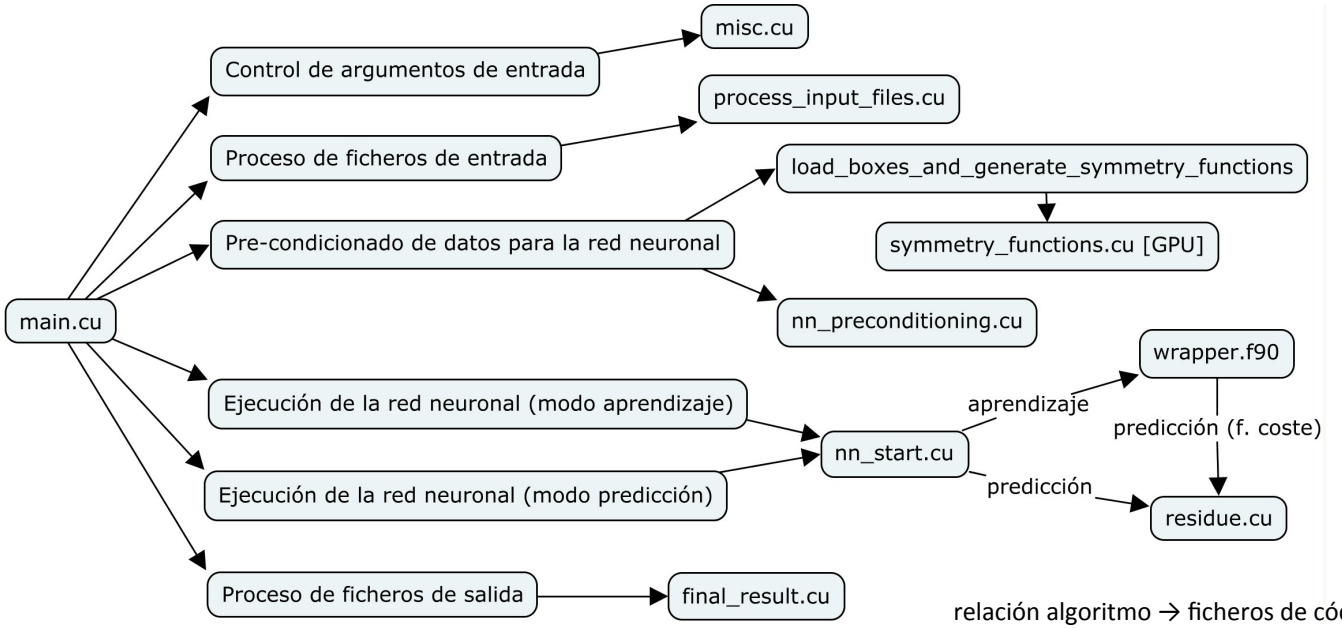
- El objetivo de este módulo es calcular el error medio de predicción y generar las gráficas que indiquen la calidad del aprendizaje y de la predicción. Por lo tanto, mediante estos datos será posible comprobar la precisión de la predicción y la calidad ajuste de la red neuronal.
- Concretamente, se imprimirá por pantalla el error medio de predicción y se crearán los ficheros “data/LEARNING_PROCESS.png” y “data/FINAL_PREDICTION.png (requiere tener instalado GNUplot).

# 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.
- 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:
  - 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.
  - El fichero del adaptador "wrapper.f90" no tiene fichero de cabecera.
  - 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.
  - 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.
  - 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.
  - 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.
  - 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.
  - Se crea un fichero .SH con ejemplos de ejecuciones de la aplicación.
  - 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.

# 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.

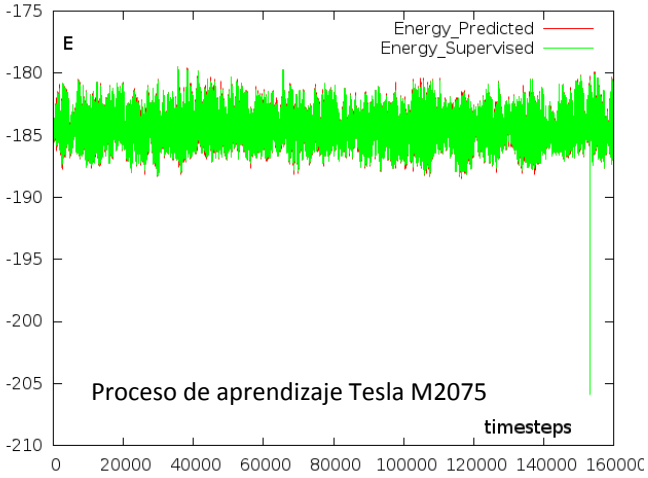


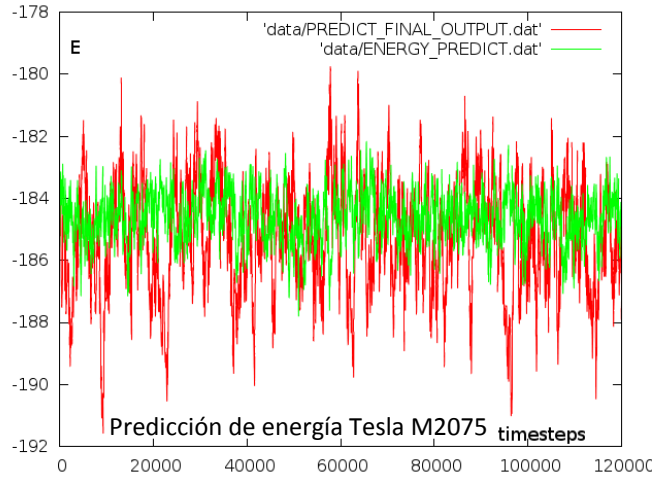

relación algoritmo → ficheros de código.

# PRUEBAS Y SIMULACIONES

- 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.
- 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:
  - Número de valores de representación por átomo = 30.
  - Número de neuronas en cada capa oculta de la red neuronal = 4.
  - Número máximo de llamadas a la función de coste = 80000.
  - 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:

# Dos conjuntos de aprendizaje (673K y 973K) y uno de predicción (773K)

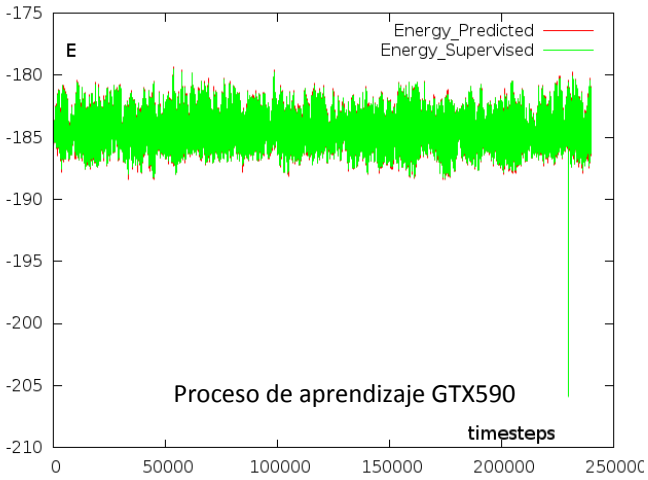


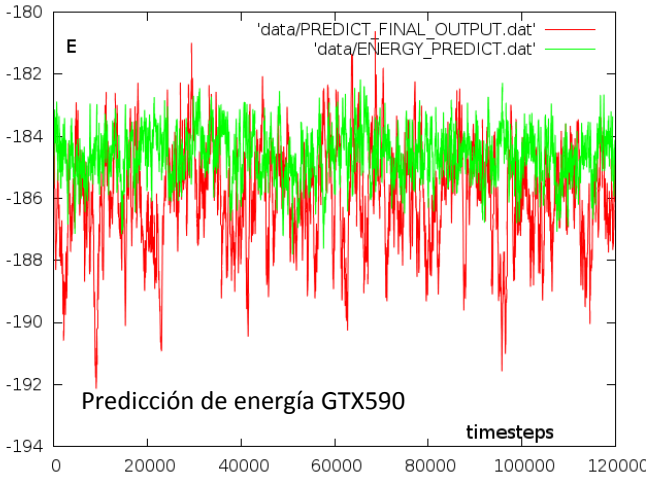




| 2-Aprendizaje | Tiempo (m) | Ē de predicción |
|---|---|---|
| GTX 590 | 740 | 0.395462 |
| GTX 660 | 755 | 0.527924 |
| GTX 960 | 383 | 0.408463 |
| GTX 980 | 340 | 0.553131 |
| Tesla M2075 | 435 | 0.720734 |

- Por un lado, como se puede observar 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 segunda gráfica 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 tabla inferior 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 lo que explica el mal ajuste representado en la predicción de la energía.
- 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).

# Tres conjuntos de aprendizaje (673K, 723K y 973K) y uno de predicción (773K)

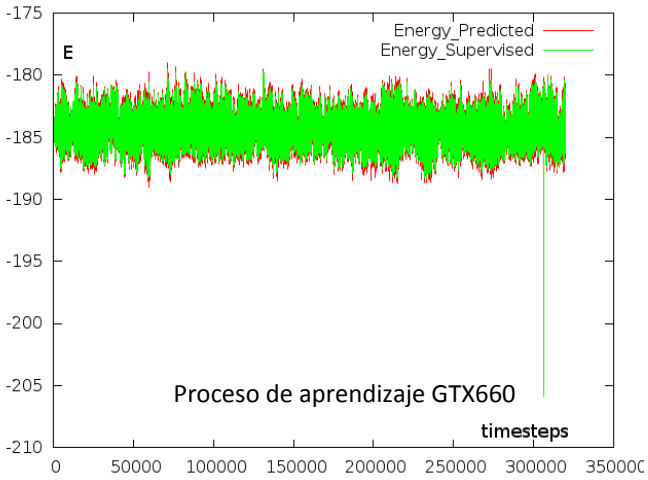


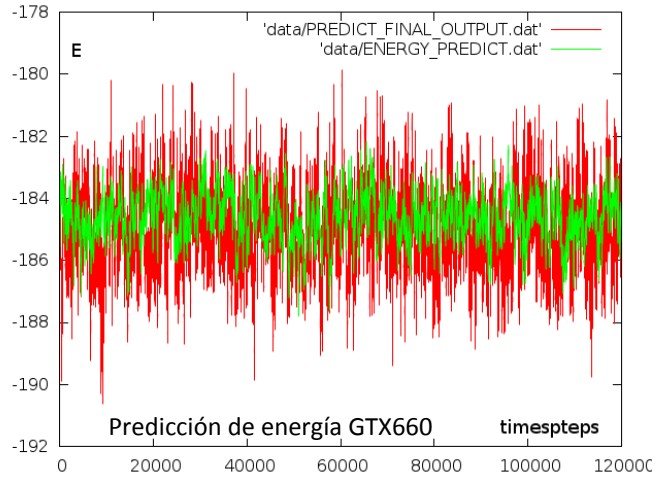




| 3-Aprendizaje | Tiempo (m) | Ē de predicción |
|---|---|---|
| GTX 590 | 1087 | 1.399277 |
| GTX 660 | 1034 | 1.256638 |
| GTX 960 | 488 | 1.067123 |
| GTX 980 | 307 | 1.328995 |
| Tesla M2075 | 576 | 0.819427 |

- Por un lado, como se puede observar 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 segunda gráfica 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 tabla inferior 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 (incluso más que en el caso anterior) lo que explica el mal ajuste representado en la predicción de la energía.
- 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).

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

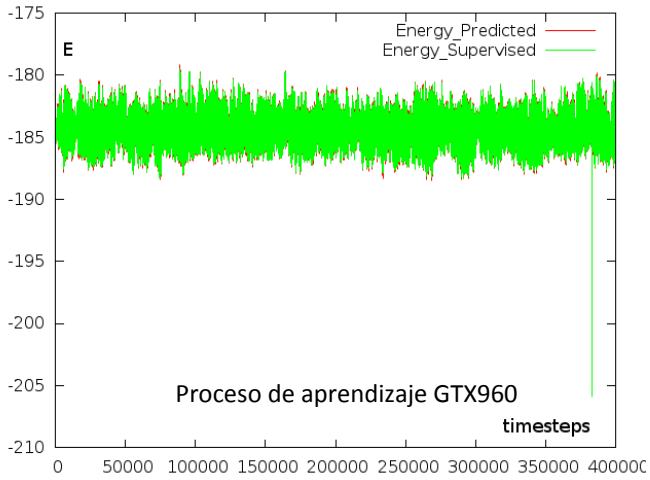


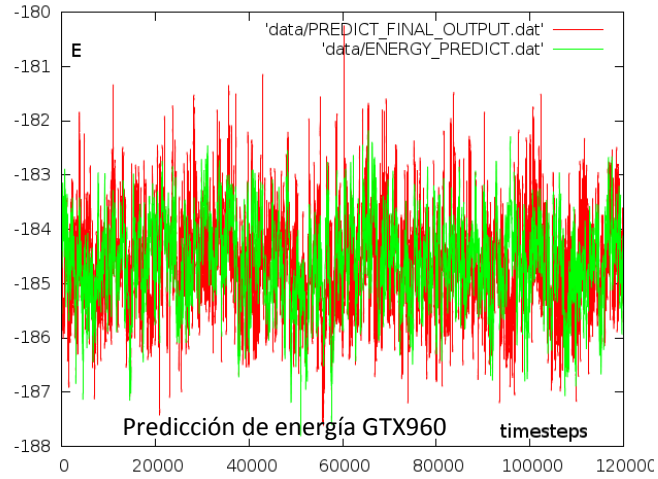




| 4-Aprendizaje | Tiempo (m) | Ē de predicción |
|---|---|---|
| GTX 590 | 1322 | 0.320465 |
| GTX 660 | 1399 | 0.396881 |
| GTX 960 | 763 | 0.638229 |
| GTX 980 | 409 | 0.631714 |
| Tesla M2075 | 732 | 0.707733 |

- Por un lado, como se puede observar 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 segunda gráfica 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 tabla inferior 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 segunda figura.
- 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).

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



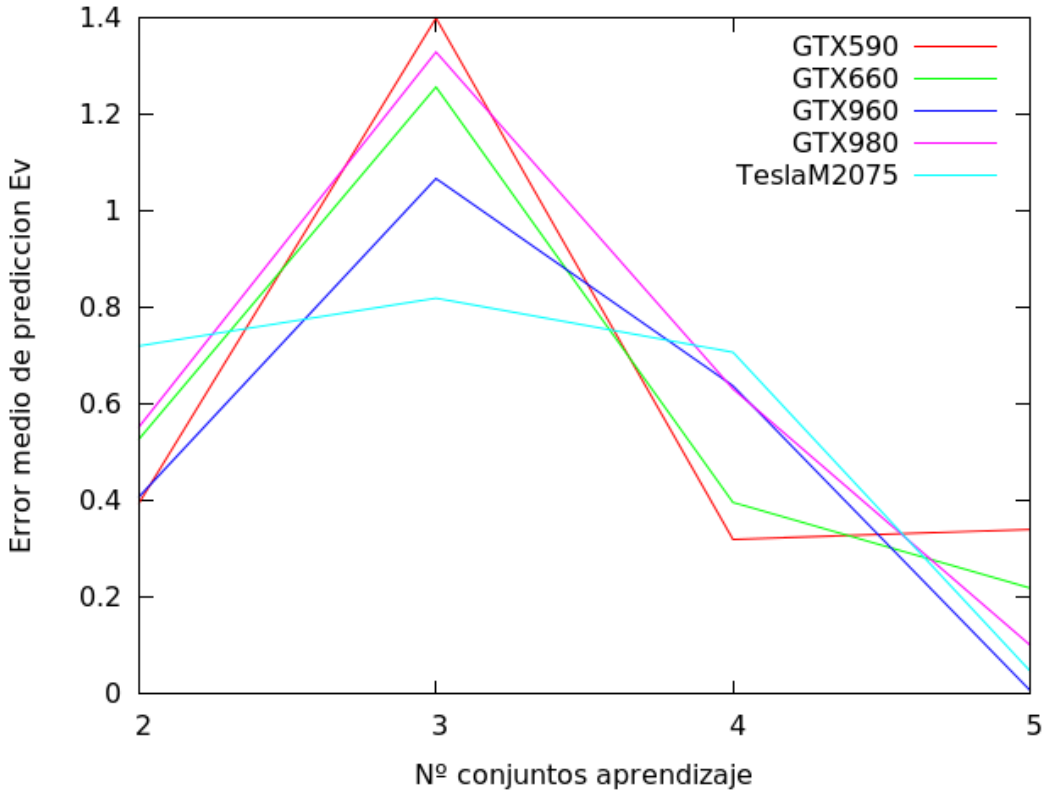




| 5-Aprendizaje | Tiempo (m) | Ē de predicción |
|---|---|---|
| GTX 590 | 1371 | 0.340820 |
| GTX 660 | 1726 | 0.219604 |
| GTX 960 | 935 | 0.007904 |
| GTX 980 | 603 | 0.101883 |
| Tesla M2075 | 1019 | 0.048141 |

- Por un lado, como se puede observar 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 segunda gráfica 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 tabla inferior 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 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.

# CONCLUSIONES

- Error medio de predicción:
  - En la gráfica 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

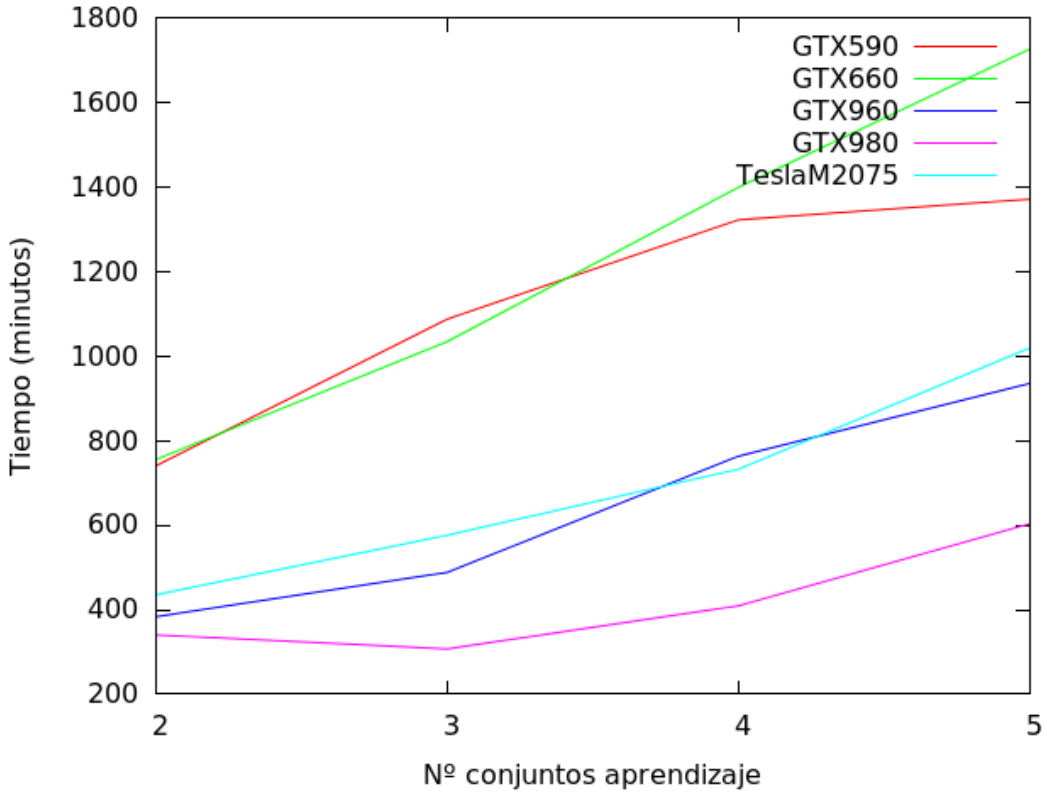



Error medio de predicción $\bar{E}$ (eV) por cada GPU.

# CONCLUSIONES

- Tiempo:
  - En la gráfica 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.

tiempo necesitado para realizar la simulación por cada GPU.



# CONCLUSIONES

- Finalmente, se concluye que:
  - La red neuronal aprende correctamente en todas las situaciones.
  - 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.
  - 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).
- 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:
    - 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.
    - 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.

# 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.