Parte III

Anexos

Anexo 1. Función coste

Como hemos comentado, existen muchas elecciones posibles para la función coste, que deben cumplir una serie de características. En el caso más sencillo podemos definir una función de coste cuadrática:

$$C(w,b) = \frac{1}{M}\sum_{x=1}^{M} C_x(w,b) = \frac{1}{2M}\sum_{x=1}^{M} ||\mathbf{y}(x) - \mathbf{a^R}(x,w,b)||^2 \tag{1}$$

donde $x \in 1, ..., M$ hace referencia a las configuraciones de entrenamiento de la red. Es importante definir el coste total como la suma normalizada de costes de todas las configuraciones para poder aplicar el *stochastic gradient descent*. Por otro lado, conviene que sea una función directa de los parámetros de la última capa de neuronas para poder aplicar el algoritmo de *backpropagation*, criterio que se cumple al ser función del output $\mathbf{a^R}(w,b)$. Su valor debe ser siempre positivo, y es conveniente que en caso de que el output de la red y el esperado sean iguales, además de tener un mínimo en la función, el valor de $C(w,b)$ sea 0. Por último, una buena función coste debe ser continua y derivable para poder determinar correctamente los cambios en los parámetros necesarios para minimizarla. La función de coste cuadrática cumple todas las características anteriores, al igual que la *cross entropy*:

$$C = -\frac{1}{M}\sum_{x=1}^{M} [\mathbf{y}\cdot \ln \mathbf{a} + (1-\mathbf{y})\cdot \ln(1-\mathbf{a})] \tag{2}$$

Por brevedad hemos omitido los índices. Aparentemente la *cross entropy* no presenta ninguna ventaja frente a la función cuadrática. En realidad, la conveniencia de esta función nace de la aplicación del algoritmo de *backpropagation*, al acelerar en gran medida el aprendizaje en las primeras etapas cuando el *output* de la red dista mucho del valor real. Es importante notar que en el caso de la función *cross entropy* los *outputs* deben estar limitados entre 0 y 1, lo que hace difícil su aplicación para problemas que no sean de clasificación.

Respecto al término de regularización, los más utilizados son los correspondientes a la regularización L1, que utiliza simplemente la norma del vector de parámetros, y la regularización L2, que utiliza la norma al cuadrado. Nosotros utilizaremos esta última, aunque no existen grandes diferencias entre el comportamiento de ambas.

$$L1(w,b) = \frac{\lambda}{M}\sum_{\nu=w,b} |\nu| \qquad\qquad L2(w,b) = \frac{\lambda}{2M}\sum_{\nu=w,b} \nu^2 \tag{3}$$

La suma se da sobre todos los pesos y *biases* de la red, y se divide por el número de configuraciones de entrenamiento M. Como ya hemos comentado, el parámetro λ se conoce como "parámetro de regularización", y es un hiperparámetro de la red. La suma sobre w, b indica que se da sobre todos los pesos y *biases*. Esto no es estrictamente necesario, puesto que se ha comprobado empíricamente que renormalizar los *biases* no produce una mejora significativa del rendimiento,

principalmente debido a que se ven mucho menos influidos que los pesos ante cambios en el input $\mathbf{x}$ y que valores elevados de éstos en ocasiones son útiles para saturar ciertas neuronas. Por esta razón, nosotros hemos optado por realizar la suma únicamente sobre los pesos. Finalmente, la función coste utilizada en nuestro trabajo resulta:

$$C = -\frac{1}{M}\sum_{x=1}^{M}[\mathbf{y}\cdot\ln\mathbf{a} + (1-\mathbf{y})\cdot\ln(1-\mathbf{a})] + \frac{\lambda}{2M}\sum_{w} w^2 \tag{4}$$

Anexo 2. *Stochastic gradient descent*

Definida la función coste, debemos ahora desarrollar un método que nos permita minimizarla variando los parámetros de la red. Esto se logra mediante la aplicación de algoritmos de descenso de gradiente, de forma que partiendo de un conjunto de parámetros $\boldsymbol{\nu}$ (en nuestro caso los pesos y *biases*) y una función coste $C(\boldsymbol{\nu})$, y calculando el vector gradiente:

$$\nabla C(\boldsymbol{\nu}) = \left(\frac{\partial C}{\partial \nu_1}, \frac{\partial C}{\partial \nu_2}, \dots\right)^T \tag{5}$$

podemos alcanzar el mínimo de la función actualizando progresivamente los parámetros según:

$$\boldsymbol{\nu} \to \boldsymbol{\nu'} = \boldsymbol{\nu} - \eta\nabla C \tag{6}$$

El cálculo del gradiente de la función coste respecto de cada parámetro se realiza mediante el algoritmo de *backpropagation*, que se detalla en el anexo 3. El factor η es un hiperparámetro de la red, se conoce como *learning rate* en el campo de la inteligencia artificial y es siempre positivo. Un valor elevado acelera el proceso de aprendizaje, pero un desplazamiento mayor en el espacio de fases en cada actualización puede provocar que el sistema sea incapaz de encontrar el mínimo o incluso diverja. En cambio, un valor demasiado pequeño alarga el proceso de entrenamiento y puede provocar que el sistema quede anclado en mínimos locales. Nos interesa por tanto un valor intermedio que evite ambos comportamientos perjudiciales, o incluso que éste vaya variando con el tiempo para ajustarse a las necesidades del entrenamiento.

Es importante notar que nuestra función coste se construye como la media de costes individuales para cada configuración de entrenamiento, y por tanto podemos expresar el gradiente como:

$$\nabla C = \frac{1}{M}\sum_{x=1}^{M}\nabla C_x \tag{7}$$

Esto implica que para actualizar los parámetros en cada paso temporal debemos calcular el gradiente de la función coste de cada configuración y promediarlos. En caso de contar con varias decenas de miles de configuraciones y cientos de miles de parámetros, el proceso puede prolongarse mucho en el tiempo hasta hacer inviable el entrenamiento. Para sortear este problema se introduce el descenso de gradiente estocástico, cuya idea principal consiste en utilizar un subconjunto de m configuraciones de entrenamiento para calcular el gradiente. Dicho subconjunto debe

ser pequeño para acelerar el proceso, pero lo suficientemente grande como para poder asumir que se cumple:

$$\frac{1}{m}\sum_{x=1}^{m}\nabla C_x \approx \frac{1}{M}\sum_{x=1}^{M}\nabla C_x = \nabla C \tag{8}$$

Es decir, suponemos que con un conjunto relativamente pequeño de configuraciones logramos estimar correctamente la dirección que toma el gradiente en el espacio de fases, acelerando el proceso en un factor M/m. En la práctica se agrupan las configuraciones de manera aleatoria en pequeños grupos llamados *mini-batches*, y se van actualizando los parámetros hasta utilizar cada uno de estos conjuntos. Cada ciclo en el que se utilizan todas las configuraciones de entrenamiento se conoce como *época*.

Anexo 3. *Backpropagation*

En este apartado vamos a estudiar cómo calcular explícitamente el gradiente de la función coste respecto de todos los pesos y *biases*. Una aproximación al problema que determine la dirección de dicho gradiente variando un sólo parámetro ligeramente y comprobando si el coste aumenta o disminuye no es factible por la inmensa cantidad de parámetros y el tiempo necesario para recalcular cada coste. Debemos por tanto ceñirnos al cálculo analítico de las derivadas, aunque a primera vista parece un problema inabarcable en redes relativamente profundas. Históricamente se han desarrollado multitud de estrategias para enfrentar esta cuestión, entre las cuales destaca el algoritmo de *backpropagation* [3], que se ha convertido en un estándar por su simpleza y su buen funcionamiento desde que fue propuesto en 1986.

El algoritmo se basa en la definición una variable vectorial $\boldsymbol{\delta^r}$ para cada capa r, denominada error, que cuantifica el efecto del input de la misma sobre la función coste según:

$$\delta_j^r \equiv \frac{\partial C}{\partial z_j^r} \tag{9}$$

donde j denota la j-ésima neurona de la capa. Nótese que hemos definido el error como el cambio ante el input z_j^l, y no el output a_j^l. En principio cualquiera de las dos versiones es válida, pero la segunda complica un poco más el álgebra. Con esta variable podemos expresar las cuatro ecuaciones fundamentales del algoritmo de *backpropagation*.

$$\delta_j^R = \frac{\partial C}{\partial a_j^R} f'(z_j^R) \tag{10}$$

$$\delta_j^r = \sum_{k=1}^{N} w_{kj}^{r+1}\delta_k^{r+1} f'(z_j^r) \tag{11}$$

$$\frac{\partial C}{\partial b_j^r} = \delta_j^r \tag{12}$$

$$\frac{\partial C}{\partial w_{jk}^r} = a_k^{r-1}\delta_j^r + \frac{\lambda}{M} w_{jk}^r \tag{13}$$

La estrategia para calcular los gradientes es sencilla: la primera ecuación nos dice cómo calcular el error de la última capa, y utilizamos la segunda ecuación para propagar los errores hacia capas cada vez más internas, lo que da su nombre al método. Las dos últimas ecuaciones nos dicen cómo determinar a partir del error de cada capa las derivadas parciales que nos interesan para minimizar el coste. La demostración de estas relaciones es inmediata a partir de la definición del error en la ecuación (9) y de la definición de z_j^r dada en el trabajo, aplicando la regla de la cadena. La función $f(z)$ es la correspondiente a la neurona con la que estemos trabajando y su derivada es conocida, al igual que la derivada de la función coste con respecto del output de la red. En el caso concreto en el que utilicemos una función coste de entropía cruzada, y neuronas sigmoid, basta sustituir

$$\frac{\partial C}{\partial a_j^R} = -\frac{1}{M}\sum_{x=1}^{M}\left[\frac{y_j}{a_j^R} + \frac{1-y_j}{1-a_j^R}\right] \tag{14}$$

$$f'(z_j^r) = \sigma'(z_j^r) = \sigma(z_j^r)\left(1-\sigma(z_j^r)\right) \tag{15}$$

para obtener las expresiones explícitas. Como comentario, nótese que las posibles divergencias en la primera de las ecuaciones anteriores por el término a_j^R en los denominadores no son tales: si el valor esperado y_j es nulo, entonces el primer término siempre es 0 y el segundo no. Si el valor esperado es 1, ocurre lo mismo con el segundo término. En resumen, ambas funciones son analíticas y se pueden calcular con facilidad conociendo el output a_j^R de la red. A partir de estas, el cálculo del resto de errores y el gradiente se realiza de forma muy rápida propagando estos resultados hacia atrás en un único paso. Hecho esto para todas las configuraciones del *mini-batch*, se calcula el gradiente final y se actualizan pesos y *biases* para disminuir el valor de la función coste.

Anexo 4. Neuronas

En el trabajo hemos explicado la neurona *sigmoid*, la más común desde los inicios de la Inteligencia Artificial hasta la aparición de las neuronas *ReLU*, que solucionaban gran parte de sus problemas asociados. En la actualidad existen una gran cantidad de neuronas posibles, cada una con sus ventajas e inconvenientes. En este anexo presentaremos algunas de ellas, incluidas todas las utilizadas en el trabajo.

La neurona más simple que podemos implementar en nuestra IA es el *perceptrón*, caracterizado por una función escalón:

$$output = \begin{cases} 0 & si \quad \sum_j w_j x_j + b < 0 \\ 1 & si \quad \sum_j w_j x_j + b \geq 0 \end{cases} \tag{16}$$

El *bias* b actúa como una umbral a partir del cual se da o no output en la neurona. Se puede demostrar que este modelo extremadamente simple es capaz de reproducir el comportamiento de una puerta lógica NAND, y resulta por tanto un conjunto funcionalmente completo, capaz de describir cualquier función lógica. No obstante, pequeños cambios en el input o en los parámetros internos del sistema pueden provocar grandes cambios repentinos en el output o no variarlo en

absoluto, dificultando el proceso de aprendizaje de la red y desaconsejando el uso de este tipo de neuronas.

Las neuronas *sigmoid* no presentan los problemas anteriores, y permiten construir redes neuronales capaces de llevar a cabo buenas predicciones, sobre todo en problemas de clasificación debido a su *output* limitado. Este tipo de neuronas tienen un mejor desempeño en redes no muy profundas, con pocas capas intermedias. La razón para ello se deriva de forma inmediata del algoritmo de *backpropagation*: vemos en la ecuación (11) que el cálculo de δ, de la que depende de forma directa el gradiente de la función coste, viene multiplicado por la derivada de la función correspondiente a la neurona. En el caso de la *sigmoid* esta derivada es nula para valores de *input* mucho mayores o mucho menores que 0, y además toma siempre un valor menor que 1. Dado que el cálculo de una δ hace uso de las δ de todas las capas posteriores, y en cada capa el valor se ve disminuido al multiplicarse por un factor menor que 1 (y en ocasiones muy pequeño), el método termina perdiendo validez para capas relativamente profundas al anularse este valor. Esto se conoce en la literatura como el problema del *vanishing gradient*.

Otr tipo de función que se utiliza principalmente para la capa de salida de redes neuronales de clasificación es la *softmax*. Su característica más importante es que la suma de los *outputs* está normalizada a 1, de forma que podemos entenderla como una distribución de probabilidad que en un problema de clasificación nos dice cuánto de probable es que el *input* de la red pertenezca a una categoría u otra. Su función resulta por tanto un poco más complicada que la anterior:

$$a_j^R = \frac{e^{z_j^R}}{\sum_k e^{z_k^R}} \tag{17}$$

donde a_j^R representa el *output* de la neurona j-ésima de la capa de salida. Su comportamiento depende así de los inputs del resto de neuronas. Vemos un ejemplo con dos neuronas a la salida en la figura (1b), suponiendo que uno de los *inputs* toma el valor $z_1 = 1{,}0$.

Para resolver el problema del *vanishing gradient* se introdujeron las neuronas $ReLU$, que obedecen la ecuación:

$$output = \begin{cases} 0 & si \quad z = \sum_j w_j x_j + b \leq 0 \\ z & si \quad z = \sum_j w_j x_j + b > 0 \end{cases} \tag{18}$$

Podemos resumir su comportamiento en $output = max\{0, z\}$. Es evidente que la derivada de esta función es 1 para todo $z > 0$. Su *output* está contenido únicamente para valores negativos, un comportamiento que puede resultar adecuado para ser utilizadas como capa de salida de la red en ciertos casos, como la predicción de la longitud de correlación que hemos estudiado en el trabajo. Durante la última década esta neurona se ha impuesto frente al resto impulsando el desarrollo de las redes neuronales profundas, y en la actualidad es la neurona más utilizada en el campo de la inteligencia artificial.

No obstante, las neuronas $ReLU$ tienen un inconveniente que en ocasiones puede perjudicar al funcionamiento de la redes. Cuando su *input* es negativo las neuronas se encuentran apagadas, su derivada es nula, y por tanto su *output* no varía ante pequeños cambios en los parámetros. Esto provoca que dependiendo de la inicialización de los parámetros o el proceso de entrenamiento, cierta cantidad de neuronas pueden quedar fijadas con *inputs* negativos resultando inútiles para

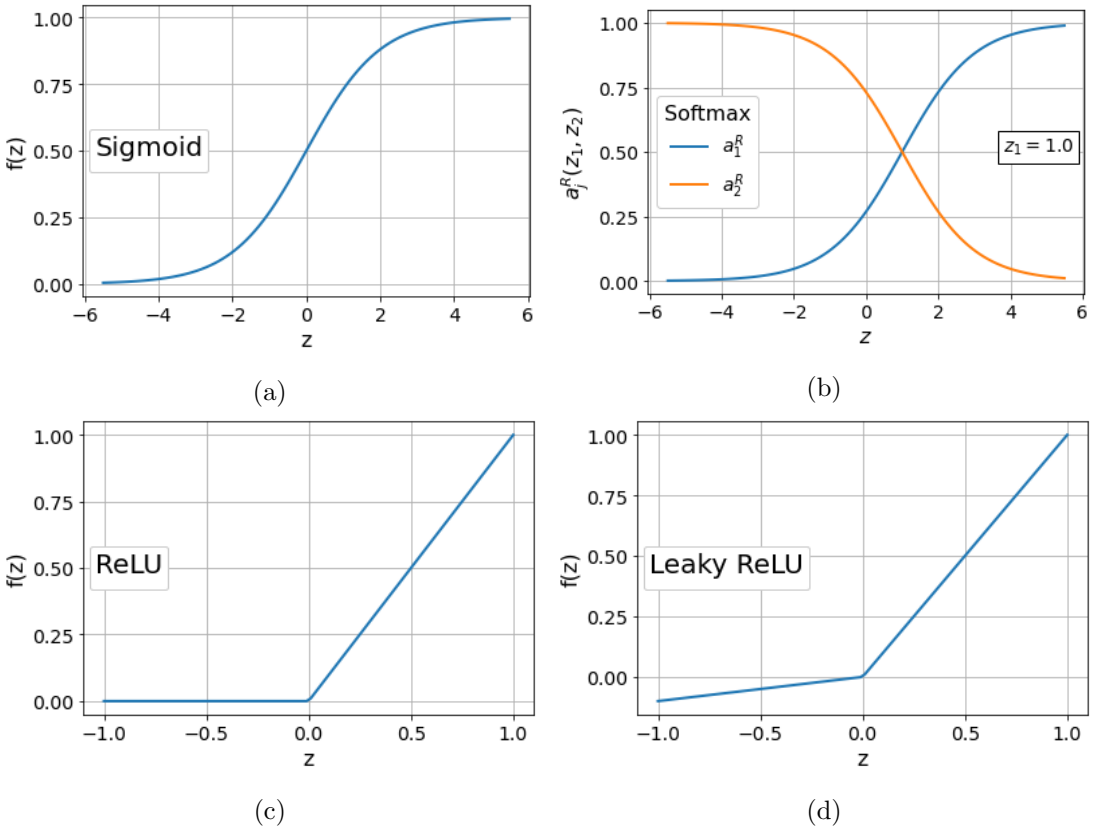


Figura 1: Representación del comportamiento de las principales neuronas utilizadas durante el trabajo. En (b) se muestra el caso de la función *softmax* aplicada al caso concreto de una capa de salida con dos neuronas, en el que uno de los *inputs* toma el valor $z_1 = 1{,}0$.

la red. Si este comportamiento se da en una cantidad suficiente de neuronas, se puede llegar a bloquear el proceso de entrenamiento imposibilitando el aprendizaje. Este es el denominado problema de las *dying ReLU*.

La solución en este caso resulta trivial, basta con añadir cierta pendiente al resultado cuando el input es negativo, de forma que la derivada de la función no se anule y el aprendizaje no se detenga. Esta nueva función se conoce como *Leaky ReLU*, y se caracteriza por un hiperparámetro α que determina la pendiente en la zona negativa.

$$output = \begin{cases} \alpha z \quad si \quad z = \sum_j w_j x_j + b \leq 0 \\ z \quad si \quad z = \sum_j w_j x_j + b > 0 \end{cases} \tag{19}$$

Es importante notar que ninguno de los problemas anteriores imposibilita el uso de cualquiera de estas funciones en redes neuronales, no aparecen en todas las ocasiones e incluso pueden resultar útiles en algunos casos. La elección de unas funciones u otras responde principalmente a criterios puramente empíricos, basados en ensayo y error, en los que los problemas comentados influyen pero no son determinantes. Por ejemplo, las neuronas *ReLU* son ahora mismo dominantes sobre otras porque en diversos experimentos se ha comprobado que se comportan mejor de forma consistente. Sin embargo, la decisión de usar neuronas *Leaky ReLU* y no *ReLU* en algunas capas de las redes convolucionales presentes en el trabajo se debe únicamente a que

los resultados parecían ser en general más estables en el primer caso, pero esto ni siquiera se cumplía en todas las ejecuciones. Esto quiere decir que la elección final de todos los componentes de una red neuronal dependen en gran medida de los objetivos a alcanzar, y que no existen reglas absolutas que permitan construir desde un principio redes adecuadas. Serán necesarios avances significativos en la comprensión del funcionamiento tanto de las redes como de neuronas individuales para empezar a vislumbrar estas reglas generales que permitan asentar sobre una base sólida el campo de la Inteligencia Artificial, joven todavía.

Anexo 5. Modelo de Ising. Computación.

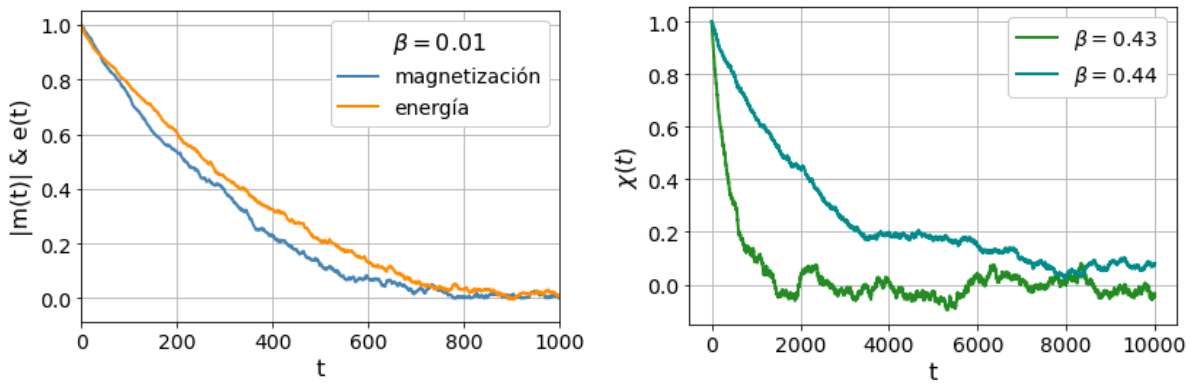
Durante este trabajo se han utilizado una gran cantidad de configuraciones de diversas redes de Ising. Para generarlas hemos utilizado principalmente códigos en C cuyas propiedades y parámetros básicos explicamos en este apartado.

Las configuraciones para cada temperatura se han generado mediante un algoritmo de Metropolis, en el que el paso de una configuración a otra depende del cociente entre las probabilidades de aparición de ambas a dicha temperatura. Es importante destacar que se usan condiciones de contorno periódicas e interacciones únicamente a primeros vecinos, tanto ferromagnéticas como antiferromagnéticas. Las cuestiones técnicas más importantes a resolver en la generación de configuraciones son la termalización y la autocorrelación entre configuraciones. Al generar las configuraciones para una nueva temperatura, debemos asegurarnos de que la distribución de probabilidad con la que aparecen corresponde realmente al nuevo caso. Para ello debemos llevar a cabo un proceso de termalización asegurando que las nuevas configuraciones corresponden verdaderamente a la nueva temperatura. En nuestro caso hemos establecido un periodo de 10000 configuraciones por defecto para todos los casos, cantidad más que suficiente para asegurar la correcta termalización del sistema incluso para las redes más grandes con $L = 64$. Por otro lado, las configuraciones que tomamos para realizar nuestros experimentos deben estar completamente descorrelacionadas entre ellas para evitar comportamientos no deseados. Esto implica que debemos desechar una gran cantidad de configuraciones intermedias entre dos tomas de datos diferentes. En nuestro caso hemos establecido que dejaremos pasar por defecto 1000 configuraciones entre cada configuración válida para el experimento, incrementando este valor hasta 10000 en el intervalo $\beta \in [0{,}43, 0{,}45]$ para combatir el *critical slowdown* propio del modelo de Ising cuando se aproxima a β_c.

Estudiamos en la figura (2a) el proceso de termalización de una red de Ising con $L = 64$, la más grande utilizada en el trabajo. Se representa tanto la magnetización como la energía por spin del modelo en la trayectoria desde una configuración completamente ordenada a una completamente desordenada, la más larga posible. Como se puede ver, en algo menos de 1000 paso temporales ambos valores alcanzan el equilibrio, y por tanto con 10000 pasos temporales entre temperaturas tenemos asegurada la termalización de la red. Por otro lado, mostramos en la figura (2b) las autocorrelaciones del modelo calculadas según la expresión:

$$\chi(t) = \frac{\langle S_i(0)S_i(t)\rangle - \langle S_i(0)\rangle^2}{\langle S_i(0)^2\rangle - \langle S_i(0)\rangle^2} \tag{20}$$

Las configuraciones se consideran descorrelacionadas cuando $\chi(t) = 0$. Es importante notar que esta expresión es válida por debajo de la temperatura de transición; por encima adquiere un valor en el equilibrio distinto de 0 para todo tiempo, por la existencia de magnetización espontánea. Esto se observa en la figura, donde se ve que cuando $\beta = 0{,}43$ las autocorrelaciones decaen exponencialmente hasta anularse en menos de 2000 pasos temporales, mientras que en $\beta = 0{,}44$, muy cerca de la transición pero por encima (recordemos que se trata de redes finitas y la transición se da antes), su valor decae a un ritmo constante hasta alcanzar $\chi(t) = 0{,}2$, a partir del cual se producen oscilaciones. Este es el umbral al que nos hemos referido antes. En definitiva, podemos concluir que 1000 pasos temporales fuera de la zona de transición, y 10000 en dicha zona, son suficientes para asegurar que las configuraciones están suficientemente descorrelacionadas.



(a) (b)

Figura 2: Resultados para una red de Ising ferromagnética con $L = 64$ (a) Magnetización y energía por spin en función del paso temporal durante un proceso de termalización. Representamos el paso de una temperatura muy baja a una muy alta. (b) Autocorrelaciones en función del tiempo, calculadas a partir de 1000 ejecuciones diferentes con la ecuación (20).

Anexo 6. Otros programas

La naturaleza computacional del presente trabajo ha hecho necesario el uso de diversos lenguajes de programación y herramientas, además de las anteriormente comentadas para la generación de configuraciones. Todas las redes neuronales utilizadas durante el trabajo se han desarrollado en *Python*. Por un lado, las redes neuronales *fully connected* se han construido a partir de los códigos incluidos en [**?**], llevando a cabo las modificaciones necesarias para adaptar el funcionamiento de redes cuyo objetivo es reconocer dígitos escritos a mano a la predicción de orden en configuraciones de Ising. Por otro lado, las redes convolucionales se han construido haciendo uso de la librería *Tensorflow*, que permite implementar redes neuronales y trabajar con ellas de manera muy sencilla. Se ha utilizado la librería *Matplotlib* para la elaboración de figuras, y los cálculos más pesados para las propiedades de las redes de Ising se han realizado en *Cython*, un lenguaje de programación que permite escribir extensiones en C con las que operar desde *Python* ganando mucha rapidez en la obtención de resultados. Se ha aplicado a cálculos

como la magnetización de las configuraciones o las correlaciones entre spines. Por último, se ha usado la función *optimize.curve_fit* de la librería *scipy* de *Python* para realizar los ajustes exponenciales que nos permiten obtener la longitud de correlación, un paso sensible porque de su correcta obtención dependen los resultados del trabajo.

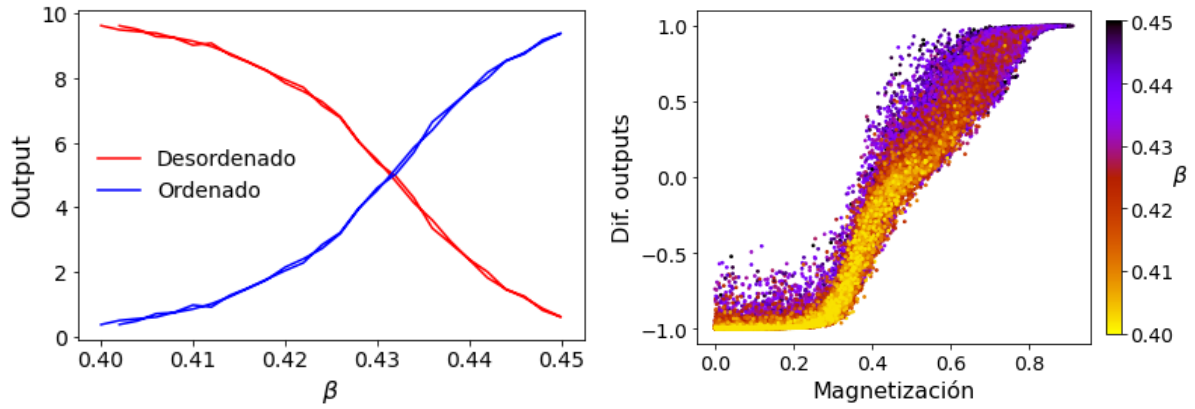
Anexo 7. Redes *fully connected* y convolucionales: consensos para la transición de fase

Hemos visto que las redes *fully connected* y las convolucionales tienen propiedades complementarias en su proceso de predicción del orden en la red de Ising. Para aprovechar lo mejor de cada una, podemos plantear un sistema en el que tomemos las predicciones por separado de varias redes y las promediemos, compensando las posibles deficiencias de redes particulares y asegurando un comportamiento consistente.

Tomamos así todas las redes neuronales utilizadas en la predicción de la transición de fase mediante clasificación de configuraciones, tanto *fully connected* como convolucionales (sección 5), y promediamos sus *outputs* para cada configuración. Utilizamos 10 redes diferentes para realizar una estimación concertada de β_c. Los resultados para una red de Ising con $L = 40$ se muestran en la figura (3). Como podemos observar, la predicción de la transición de fase tiene una forma muy similar a los casos anteriores por separado. El promediado sobre las diez predicciones suaviza las curvas y sitúa la predicción a medio camino de los resultados por separado, evidentemente. La ventaja de este método es que las configuraciones dudosas en las que las redes se encuentran divididas dan lugar a predicciones de en torno a 0,5 en ambos *outputs*, ordenado y desordenado, y por tanto influyen muy poco en la determinación de β_c.

Por otro lado, vemos un comportamiento peculiar en la figura (3b), que compara las predicciones con las magnetizaciones. Se observa un comportamiento que en la mitad inferior recuerda al de las *fully connected* y en la mitad superior, al de las convolucionales. Lo que ocurre en la zona de transición es que las redes *fully connected* pasan a considerar configuraciones ordenadas para magnetizaciones relativamente pequeñas, de en torno a 0,4, pero las convolucionales apenas han comenzado a ascender lastrando el crecimiento de la curva. Esta adquiere una forma similar a la curva de las *fully connected*, que son quienes guían el aumento. Por otro lado, a partir de magnetizaciones mayores de 0,5 las redes *fully connected* consideran todas las configuraciones ordenadas, y son las convolucionales las que guían el aumento de las predicciones, otorgando su forma a la curva en el último tramo. De esta manera obtenemos las ventajas de ambos métodos, adelantamos la magnetización a la que se produce la transición frente a las redes convolucionales, y tenemos en cuenta las relaciones entre spines, a diferencia de las redes *fully connected*. En este sentido, el sistema clasifica la configuración (6a) mostrada en el trabajo como desordenada, pero por un pequeño margen (0,533 frente a 0,467), por lo que el posible error tendrá muy poca influencia en el resultado final. Por supuesto, clasifica correctamente la segunda configuración como ordenada.

Todo esto nos permite obtener una nueva predicción para la transición de fase en el límite termodinámico repitiendo el procedimiento para distintas L. Se muestra el resultado en la figura (4). Como podemos observar, la dependencia es mucho más lineal que en los casos anteriores



(a) (b)

Figura 3: Predicciones para una red de Ising ferromagnética con $L = 40$, haciendo uso al mismo tiempo de 5 redes *fully connected* y 5 redes convolucionales. (a) Predicción de la transición de fase, que se sitúa en $\beta_c = 0{,}434$. (b) Predicción de la red frente a la magnetización de cada configuración.

y da lugar a un ajuste muy preciso que sitúa la transición en el límite termodinámico en $\beta_c = 0{,}4403 \pm 0{,}0004$. Es un valor extremadamente cercano al teórico, que además incluye en su intervalo de error. Es importante destacar además que el nuevo resultado no se sitúa a medio camino entre los resultados por separado para los distintos tipos de redes ($\beta_c = 0{,}4362 \pm 0{,}0012$ para las *fully connected* y $\beta_c = 0{,}4414 \pm 0{,}0008$ para las convolucionales), y además es mucho más cercano al obtenido con redes convolucionales. En cualquier caso se trata de un resultado muy satisfactorio que mejora los anteriores y justifica el uso de este tipo de estrategias para los problemas de clasificación, y en concreto para la predicción de la transición de fase en el modelo de Ising.

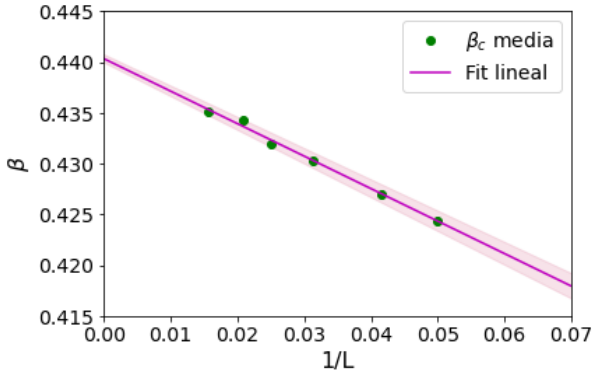


Figura 4: Predicciones de β_c para redes de Ising ferromagnéticas de diferentes tamaños ($L = 20, 24, 32, 40, 48, 64$), haciendo uso de 5 redes *fully connected* y 5 redes convolucionales. La transición en el límite termodinámico (cuando $1/L \to 0$) se sitúa en $\beta_c = 0{,}4403 \pm 0{,}0004$.

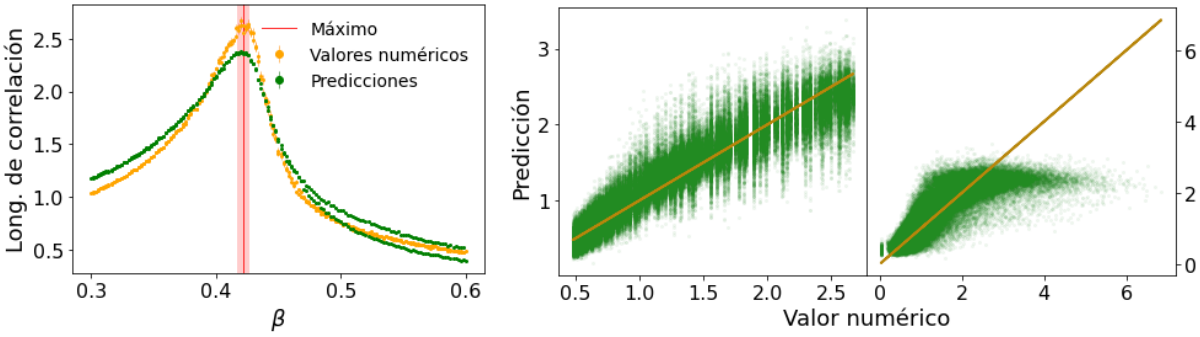
Anexo 8. ¿Por qué correlaciones?

En el trabajo hemos utilizado la longitud de correlación como método para predecir la transición de fase del modelo de Ising. No es la única opción, otros observables como el calor específico o la susceptibilidad también divergen en β_c, y podrían haber sido elecciones válidas para llevar a cabo los experimentos. Sin embargo, como hemos comentado durante el trabajo, la propiedad fundamental que inclina la balanza hacia la longitud de correlación es que al poder ser calculada usando todas las posibles parejas de spines de una configuración, podemos asignar un valor a la longitud de correlación que la red, al menos en teoría, debe ser capaz de calcular partiendo únicamente de dicha configuración.

Esto contrasta con lo que ocurriría si entrenásemos la red en la predicción de la susceptibilidad o el calor específico. A diferencia de la longitud de correlación, se obtienen a partir de promedios de la magnetización y la energía de una gran cantidad de configuraciones, y por tanto caracterizan al conjunto de configuraciones correspondientes a una temperatura, y no a cada una por separado. Esto implica que no existe una relación directa entre *input* y *output*, comprometiendo la capacidad predictiva de la red. Para resaltar el problema claramente, supongamos que entrenamos la red con configuraciones cercanas a la transición de fase. Debido a las divergencias de χ y C_v, es posible que para valores de β muy cercanos entre sí el calor específico o la susceptibilidad sean notablemente diferentes. Sin embargo, existe una probabilidad no despeciable en estos rangos de que aparezcan las mismas configuraciones en temperaturas muy próximas. Estaríamos por tanto entrenando la red con dos configuraciones iguales esperando dos *outputs* diferentes. Esto no puede ocurrir en el caso de las correlaciones tal y como las calculamos, y por ello las hemos seleccionado.

Mostramos en la figura (5) los resultados que habríamos obtenido si en lugar de utilizar la longitud de correlación calculada para cada configuración hubiéramos usado el promedio correspondiente a la temperatura a la que ha sido generada. Podemos ver que el comportamiento global es correcto, situándose el máximo en el mismo lugar que en los casos anteriores y con magnitud similar, aunque ligeramente subestimada. Vemos en la figura (5b) las predicciones de la red frente al valor calculado numéricamente. A la izquierda estimamos dicho valor numérico como la media de longitudes de todas las configuraciones correspondientes a una misma β, y a la derecha, a cada longitud de correlación por separado. Vemos que las predicciones de la red son bastante razonables si las comparamos con las medias para cada β, pero sin embargo son completamente erróneas para configuraciones individuales con longitudes de correlación abultadas. Podemos observar que la red ni siquiera es capaz de reproducir una tendencia ascendente en las predicciones para tamaños mayores de las fluctuaciones, lo que siembra dudas acerca de los criterios utilizados para realizar dichas predicciones. Por ejemplo, existe la posibilidad de que la red esté traduciendo las configuraciones a una temperatura, y de ahí asigne un valor de la longitud de correlación, subvirtiendo el objetivo de la red y utilizando la transición de fase como medio para realizar las predicciones. De esta manera invalidaría la estimación de β_c, porque la red tiene ya codificado el máximo que debe asignar a las predicciones según la temperatura que estima para cada configuración. Es importante recalcar una vez más que la relación entre configuración y longitud de correlación sí existe y es más o menos directa, simplemente la red no la ha aprendido como tal y está recurriendo a otros caminos. En definitiva, pese a los resul-

tados aparentemente buenos no podemos considerar que el funcionamiento es adecuado. Este mal funcionamiento probablemente se deba al método de entrenamiento propuesto para la red, resaltando la importancia de un correcto diseño de todas las fases implicadas en la construcción y entrenamiento de las redes neuronales.



(a) (b)

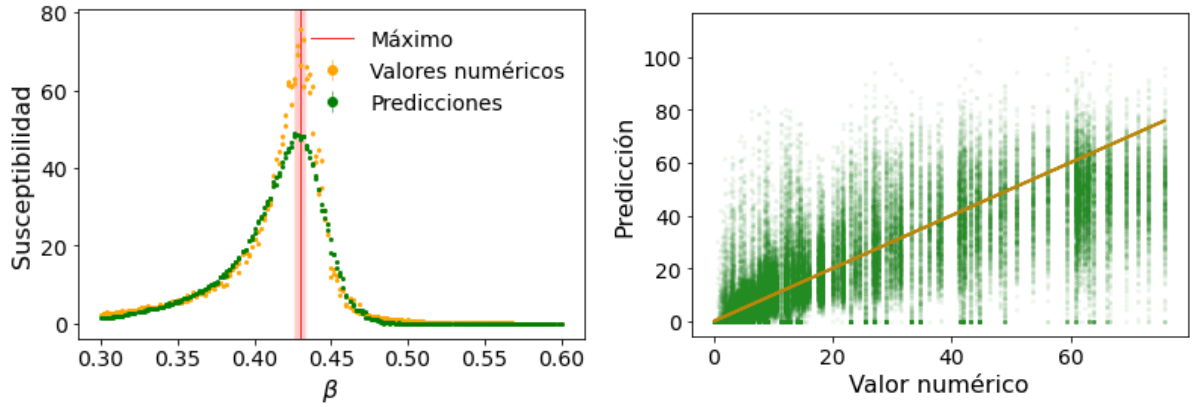
Figura 5: Predicciones para la longitud de correlación en una red de Ising ferromagnética con $L = 40$ haciendo uso de una red neuronal convolucional con una neurona $ReLU$ a la salida. (a) Predicciones y resultados numéricos para el valor medio de la longitud de correlación en función de β. El máximo se sitúa en $\beta_c = 0{,}422 \pm 0{,}04$. (b) Izquierda: predicciones que obtenemos de la red frente al valor obtenido de forma numérica promediando las longitudes de correlación para cada β. Derecha: predicciones frente a las longitudes de correlación obtenidas para cada configuración.

Otro ejemplo quizá más interesante que la longitud de correlación es la susceptibilidad magnética. A diferencia del caso anterior, en el que el valor para el entrenamiento se obtiene como media de longitudes, ahora el cálculo se debe realizar necesariamente con la influencia de todas las configuraciones pertenecientes a una misma β. Desde el punto de vista computacional podemos definir la susceptibilidad magnética como:

$$\chi(T) = V(\langle m^2 \rangle - \langle |m| \rangle^2) \tag{21}$$

donde V es el número de spines de la red, y m la magnetización de la configuración. Los promedios se realizan sobre todas las configuraciones generadas para una misma β. Se trata por tanto de una cantidad que no se puede calcular partiendo de una única configuración, y por tanto la red no puede establecer una relación matemática directa entre configuración y resultado. Vemos los resultados en la figura (6). Como podemos observar, la red neuronal logra establecer correctamente la posición del máximo de la susceptibilidad pese a desviarse notablemente en su magnitud. Vemos sin embargo a la derecha que las predicciones para cada configuración son manifiestamente mejorables extendiéndose por un amplio rango de valores para todas las β. Pese a que los promedios finales obtienen resultados decentes, la predicción para cada configuración individual no es confiable. Nos encontramos en una situación similar al caso anterior: la red se comporta correctamente haciendo aquello para lo que ha sido entrenada, pero es incapaz de trasladar este buen resultado a la configuración individual. Estas dificultades también son esperables teniendo en cuenta que la susceptibilidad no está definida por configuración, sino que caracteriza una temperatura, y teniendo en cuenta que todas las configuraciones tienen probabilidades no nulas de aparecer en todas las temperaturas, no es realista esperar un correcto

funcionamiento fuera de los promedios. En el fondo estamos entrenando a la red en la comprensión de las distribuciones de probabilidad de las configuraciones en función de la temperatura, y luego exigiendo que aplique este conocimiento para predecir a partir de una sola configuración, algo muy complicado. Esto nos lleva inevitablemente a los temores planteados en el caso de las correlaciones, es posible que la red esté prediciendo una temperatura y asignando después una susceptibilidad, invalidando los resultados para la transición de fase que podamos obtener con este método.



(a) (b)

Figura 6: Predicciones para la susceptibilidad en una red de Ising ferromagnética con $L = 40$ haciendo uso de una red neuronal convolucional con una neurona *ReLU* a la salida. (a) Predicciones y resultados numéricos para el valor medio de la susceptibilidad en función de β. El máximo se sitúa en $\beta_c = 0{,}43 \pm 0{,}03$. (b) Predicciones de la red frente al valor numérico calculado para cada β. Cada una de las líneas verticales corresponde a todas las configuraciones generadas para una β, a la que se le asigna una única susceptibilidad.

Vemos así la importancia de poder establecer relaciones claras entre datos de entrenamiento y predicciones, al igual que la asombrosa capacidad de las redes neuronales para adaptarse y dar respuesta a casi cualquier problema. Esta capacidad puede actuar no obstante en contra de nuestros objetivos, pues como hemos comprobado en este anexo es posible que redes aparentemente bien comportadas no estén basando sus predicciones en los criterios que nosotros esperamos, dando lugar a una mala capacidad de generalización, o incluso extrayendo directamente de nuestros datos la información que esperamos que sea capaz de obtener de forma autónoma, invalidando nuestros resultados.