Facultad de Ciencias

DESPLIEGUE DE MODELOS DEEP LEARNING PARA PROCESAMIENTO DE DATOS CLIMÁTICOS

(Deployment of deep learning models for climate data processing)

Trabajo de Fin de Máster
para acceder al

MÁSTER INTERUNIVERSITARIO EN DATA SCIENCE

Autor: Sergio Soler López

Director\es: José Manuel Gutiérrez Llorente

Septiembre 2018

Agradecimientos

En primer lugar, he de agradecer al grupo de Meteorología y Minería de Datos del IFCA, y especialmente a José Manuel Gutiérrez por dirigir el proyecto y darme la oportunidad de trabajar y hacer aquí el TFM. También he de agradecer a Jorge Baño Medina toda la ayuda prestada durante este TFM y su amabilidad para resolver todo tipo de problemas y avanzar en este proyecto. Tampoco me puedo olvidar de Maialen Iturbide, que me ha ayudado con el código cuando no me salían las cosas.

Otra persona que ha sido fundamental para la realización de este TFM ha sido Aida Palacio, encargada del soporte de los servicios Cloud del IFCA, ya que siempre ha sido capaz de encontrar soluciones a los problemas técnicos que iban apareciendo de manera eficaz.

También he de dar las gracias a Jesús Marco y a todo el grupo de Computación Avanzada del IFCA, en especial a Daniel García y Fernando Aguilar, por todo su tiempo y apoyo prestados y por darme la oportunidad de trabajar con imágenes de satélite a la vez que realizaba este máster, ya que ha sido un gran complemento a la formación que se impartía en el máster y me ha servido para mejorar mis conocimientos de programación, además del hecho de ser un campo muy interesante por investigar.

Por último, es importante dar las gracias a todos los profesores del máster por los conocimientos adquiridos, a mis compañeros de clase, y a todo el IFCA en general, ya que ha sido prácticamente mi segunda casa durante todo este año y he tenido la oportunidad de trabajar de la mano de grandes profesionales en sus respectivos campos y aprender un montón de cosas.

En general he de decir que los 5 años que he pasado aquí en Cantabria han sido muy buenos, además es una tierra preciosa y un lugar muy agradable para vivir. Ahora me marcho a seguir mi formación, pero siempre tendré un buen recuerdo.

Contenido

Resumen

En este trabajo se ha validado la posibilidad de hacer downscaling estadístico con redes neuronales profundas (deep learning). Para ello se han realizado dos experimentos; en primer lugar se ha tomado como benchmark el experimento definido en el proyecto VALUE (intercomparación de métodos de downscaling sobre Europa) usando como predictores variables atmosféricas de larga escala (del reanálisis ERA-Interim) y como predictando la precipitación local en 86 estaciones sobre Europa, para el período 1979-2008. Con estos datos se ha podido hacer downscaling estadístico con redes neuronales utilizando tanto CPUs como GPUs para el aprendizaje. Como métrica se ha utilizado el ROC skill score, con un resultado de 0.76+-0.02. En cuanto a los tiempos se ha podido comprobar que en GPU ha sido cuatro veces más rápido que en CPU.

En segundo lugar se ha hecho una extensión de este experimento para comprobar su escalabilidad. Para ello se han considerado los mismos predictores, pero como predicando se ha considerado la precipitación en una rejilla regular sobre Europa (EOBS con ~50km de resolución, equivalente a 3259 “estaciones”). Con estos datos se han entrenado modelos de downscaling (solo con CPU en este caso) y se ha obtenido un ROC skill score de 0.79+-0.02; los tiempos de ejecución han aumentado unas cuatro veces respecto al primer caso con CPU también.

Como resultado final de este TFM, se ha podido demostrar que es posible hacer downscaling estadístico a escala continental, sobre Europa en nuestro caso. También se ha validado que el uso de GPU reduce los tiempos en el entrenamiento respecto a las CPU.

Palabras clave: modelos climáticos, redes neuronales, downscaling, ROC skill score, GPU.

Abstract

In this work, the possibility of performing statistical downscaling with deep neural networks has been validated. For this purpose two experiments have been carried out; in the first one we took as a benchmark the experiment defined in the VALUE Project (intercomparison of downscaling methods over Europe), using as predictors long-range atmospheric variables (from the ERA-Interim reanalysis) and as predicting the local precipitation over Europe for the period 1979-2008. With this data it has been possible to do statistical downscaling with neural networks, using both CPUs and GPUs for the learning. As a metric we used the ROC skill score, with a result of 0.76+-0.02. Regarding the times we registered that GPUs were four times faster than CPU.

In second place an extension has been done to validate the scalability. For this purpose the same predictors have been considered, but as a predicting we took the precipitation in a regular grid over Europe (EOBS with 50km resolution, equivalent to 3259 “stations”). With this data, downscaling models have been trained (only with CPU in this case) giving as a result a ROC skill score of 0.79+-0.02; the execution times have increased about four times compared to the first case with CPU as well.

As a final result of this work, it has been possible to demonstrate that it is possible to perform statistical downscaling on a continental scale, over Europe in our case. It has also been validated that the usage of GPU reduces training times compared to CPUs.

Key words: climatic models, neural networks, downscaling, ROC skill score, GPU.

Introducción

En los últimos años se está acumulando un volumen de datos en muchos ámbitos diferentes que es imposible de manejar con técnicas de análisis tradicionales. Por ello, las ciencias de datos supondrán una revolución que transformará muchas áreas de la ciencia. En particular, en el ámbito de la meteorología, se empiezan a ver aplicaciones de técnicas de deep learning en campos como el estudio de la intensidad de los ciclones, la detección de eventos extremos o el downscaling estadístico [21].

El objetivo de este trabajo de fin de máster es comprobar que es posible aplicar redes neuronales para hacer un downscaling estadístico de la precipitación utilizando distintos conjuntos de datos climáticos, y comprobar la diferencia en los tiempos de ejecución al usar GPUs y CPUs.

Este trabajo se realiza en el grupo de Meteorología y Minería de Datos del Instituto de Física de Cantabria con la colaboración del grupo de Computación Avanzada, también del mismo instituto, que ha facilitado el soporte informático de la infraestructura utilizada (en particular, las máquinas virtuales).

Desde que en las conferencias de Dartmouth en 1956 se presentara el término de inteligencia artificial [1] hasta el nuevo boom de las redes neuronales en la segunda década del siglo XXI, estás tecnologías han estado limitadas por la carencia de una ingeniería lo suficientemente avanzada para realizarlas de forma barata y eficiente. En los últimos años estas técnicas han dado un paso de gigante debido principalmente a la capacidad de las GPUs para paralelizar más rápido, más barato y más potente. Otro factor a tener en cuenta en este boom es el Big Data, ya que cada vez tenemos más datos a nuestro alcance a partir de los cuales podemos obtener información de calidad.

Cuando nos referimos a inteligencia artificial nos referimos a tecnologías que son capaces de hacer determinadas tareas específicas para las que hayan podido ser preparadas, con una eficiencia similar a la de un ser humano, o incluso mejor (por ejemplo, reconocimiento de patrones). En un nivel más avanzado de la inteligencia artificial tenemos el machine learning, que consiste fundamentalmente en analizar los datos a partir del uso de algoritmos para obtener información relevante. En lugar de ejecutar rutinas con unas instrucciones específicas se entrena una máquina con grandes volúmenes de datos para que aprendan una determinada tarea; un ejemplo de estos métodos pueden ser los árboles de clasificación y predicción, las redes bayesianas, etc. Una de las mejores aplicaciones del machine learning es el análisis de imágenes donde, para algunas tareas, las máquinas han superado ya al ser humano.

Finalmente, en un nivel superior se encuentra el deep learning (ver figura 1) con las redes neuronales. Estos modelos están inspirados en las interconexiones de las neuronas de nuestros cerebros, organizadas en una serie de capas con conexiones que se aprenden a partir de los datos; hablaremos de todo esto con mucho más detalle en la sección de redes neuronales.

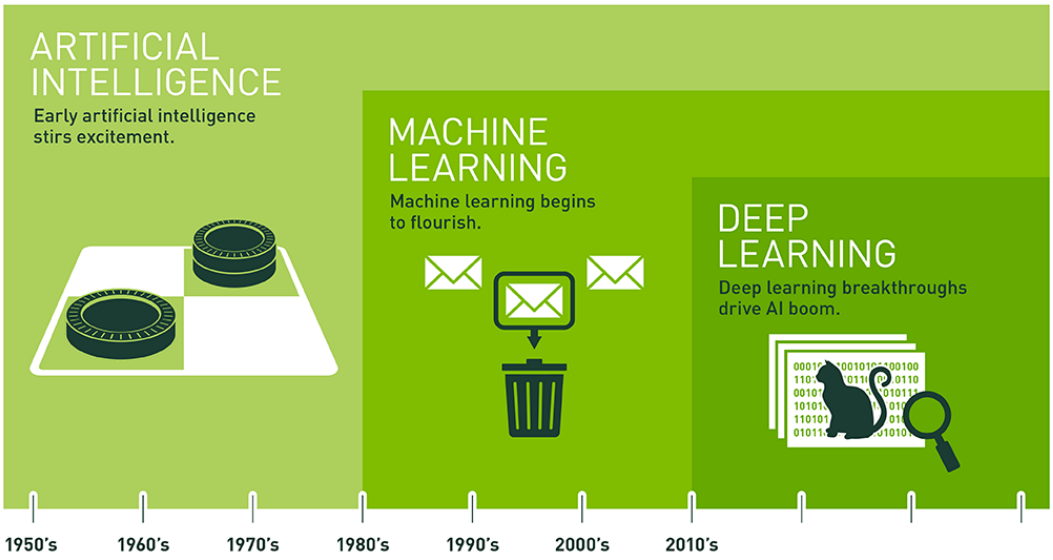


Figura 1: Evolución temporal de la inteligencia artificial [1].

El capítulo 1 de este trabajo de fin de máster consta principalmente de 3 secciones: en la primera se hace una introducción al modelado del clima, en la segunda se habla de downscaling estadístico y en la tercera de redes neuronales. En el capítulo 2 nos centramos en describir los experimentos que queremos utilizar, así como el espacio de trabajo. A continuación, en el capítulo 3 exponemos y discutimos los resultados obtenidos. Finalmente, en el capítulo 4, concluimos exponiendo los resultados más relevantes de este trabajo e indicamos algunas propuestas para el futuro.

Capítulo 1: Fundamentos

Sección 1.1: Modelado del clima

Un modelo climático global (GCM) se puede definir como una representación matemática del sistema climático basada en principios físicos, químicos y biológicos. Las ecuaciones de estos modelos se resuelven numéricamente, lo que implica que las soluciones que proporcionan son discretas en espacio y tiempo, por lo que los resultados representan promedios sobre una cuadrícula que depende de la resolución del modelo. La resolución espacial de los GCM suele rondar los cientos de km, y se utilizan para distintas escalas temporales de predicción desde días (para la predicción meteorológica a corto plazo) a años (para aplicaciones de cambio climático), dependiendo del aspecto que se quiera estudiar.

Las cuadrículas de la malla de los GCMs son demasiado grandes para representar fenómenos a pequeña escala como la turbulencia, las nubes, las tormentas, elementos topográficos, etc. Como consecuencia de ello, es necesario hacer parametrizaciones basadas en evidencias o argumentos teóricos para relacionar los elementos a gran escala con los elementos a pequeña escala, estas parametrizaciones son a menudo fuente de error porque solo son capaces de describir fenómenos de primer orden, y no en todas las condiciones de contorno. Además, los modelos citados anteriormente requieren algún input procedente de observaciones directas, tanto para condiciones de contorno como la topografía del terreno, la batimetría del mar, o propiedades de los suelos, como forzamientos externos como la irradiancia solar, o la concentración de CO_2 en la atmósfera.

Dependiendo de la dimensión de la zona de estudio podemos distinguir tres tipos de escalas.

Fenómenos a gran escala

Nos referimos a configuraciones que tienen lugar, como su nombre indica, a escalas muy grandes de cientos a miles de kilómetros. Probablemente uno de los fenómenos a gran escala más conocidos sea el Niño, en el cual se produce un incremento de la temperatura del agua en el pacífico cerca de las costas de Perú (ver figura 2), que es capaz de alterar el clima en muchas otras regiones del planeta muy alejadas. El Niño es un fenómeno cíclico que se repite cada 3-8 años aproximadamente que se alterna con la Niña, que es la situación inversa, es decir, un enfriamiento de las aguas.

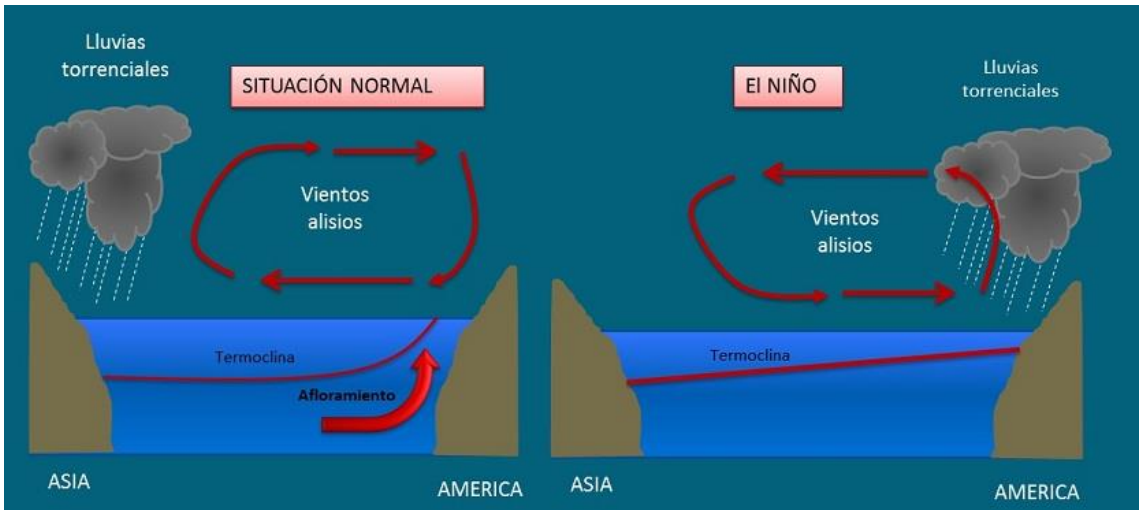


Figura 2: Esquema de la situación de la atmósfera y el océano durante el niño [5].

Otro fenómeno a gran escala importante a la hora de predecir el tiempo en Europa es la Oscilación del Atlántico Norte o NAO (ver figura 3); este efecto tiene una gran influencia sobre el clima de Europa y la costa este de los Estados Unidos; durante una NAO positiva las bajas presiones se encuentran en el Atlántico Norte, cerca de Islandia, las borrascas circulan por el norte de Europa y el anticiclón se queda semiestacionario en las Azores; la situación inversa es una NAO negativa, donde las tormentas son frecuentes en el sur.

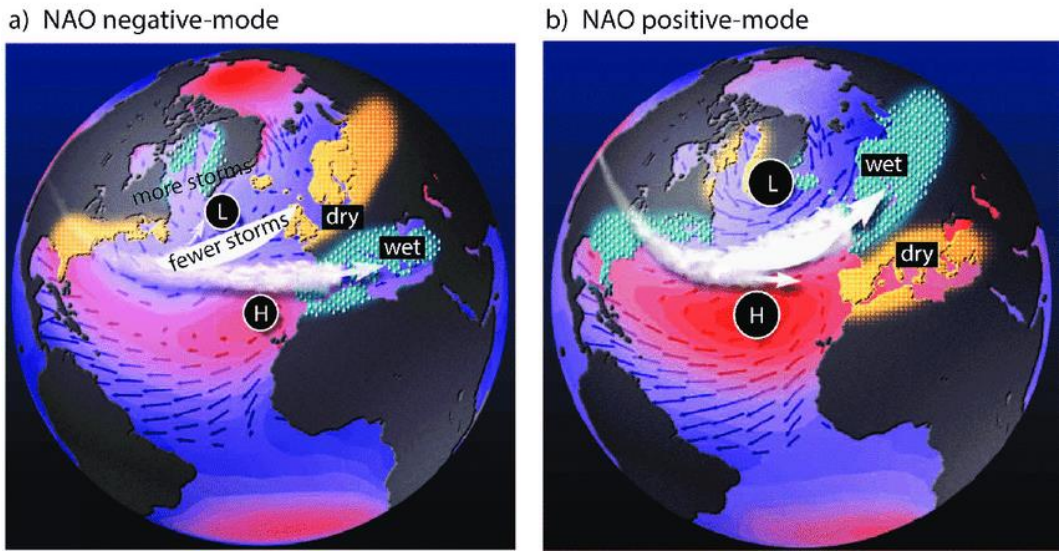


Figura 3: Comparación de una situación con NAO+ y NAO- [6].

Fenómenos de escala regional

Los fenómenos de escala regional son fenómenos que ocupan una extensión considerablemente más pequeña que los de escala global, entre decenas o pocos cientos de kilómetros. Dentro de estos fenómenos podemos poner como ejemplos las tormentas, brisas marinas y algunos tipos de viento, por ejemplo, el viento sur que eleva considerablemente las temperaturas en Cantabria y País Vasco.

Fenómenos de escala local

Hace referencia a las condiciones en una determinada localidad; por ejemplo, si tenemos un anemómetro situado en la cima de una montaña registra vientos muy superiores que si lo situamos en el fondo del valle.

Tipos de modelos

Cuando se diseñan modelos climáticos es obligatorio hacer simplificaciones dependiendo de los fenómenos que se quieran estudiar y de la escala temporal. Lo primero que hay que decidir es qué variables o procesos van a ser tomados en cuenta. Para aplicaciones de escala climática (por ejemplo, estudios de cambio climático) los modelos climáticos han de representar, al menos, el comportamiento de la atmósfera, de los océanos y del hielo marino. Aunque también se puede representar otros aspectos como el ciclo del carbono o la vegetación.

En cuanto a la complejidad de los procesos incluidos, los GCMs son los modelos más completos y tratan de tener en cuenta todas las propiedades importantes del sistema. Por otra parte, los EBMs son modelos de balance de energía con una alta simplificación de la dinámica del sistema climático. Otro tipo de modelos que podemos citar son los EMIC, modelos de complejidad intermedia, que son como un término intermedio entre los EBM y GCM. En este trabajo nos centraremos en los GCMs.

Componentes de un modelo climático

Atmósfera

Las ecuaciones básicas que rigen el comportamiento de la atmósfera se pueden formular como un conjunto de ecuaciones con siete incógnitas: la velocidad (x, y, z), la presión p, la temperatura T, la humedad específica q y la densidad ρ [7]. Las ecuaciones son las siguientes:

La segunda ley de Newton

$$\frac{d\vec{v}}{dt} = -\frac{1}{\rho}\vec{\nabla}p - \vec{g} + \vec{F}_{fric} - 2\vec{\Omega}\, x\, \vec{v} \quad (1)$$

En esta ecuación d es la derivada total, con el término de transporte incluido,

$$\frac{d}{dt} = \frac{\partial}{\partial t} + \vec{v}\vec{\nabla} \quad (2)$$

Donde **g** es el vector de gravedad aparente (considerando la fuerza centrífuga), $\mathbf{F_{fric}}$ es la fuerza de fricción, y **Ω** es el vector de la velocidad angular de la Tierra (el último término es la fuerza de Coriolis).

La ecuación de conservación de la masa

$$\frac{\partial \rho}{\partial t} = -\vec{\nabla}(\rho\vec{v}) \quad (3)$$

La conservación de la masa de vapor de agua

$$\frac{\partial \rho q}{\partial t} = -\vec{\nabla}(\rho\vec{v}q) + \rho(E - C) \quad (4)$$

Donde E y C son la evaporación y la condensación respectivamente.

Conservación de la energía (primera ley de la termodinámica)

$$Q = C_P \frac{dT}{dt} - \frac{1}{\rho}\frac{dp}{dt} \quad (5)$$

Donde Q es el calor por unidad de masa y C_P es el calor específico.

La ecuación de estado

$$pV = nRT \quad (6)$$

Aún con todo, estas ecuaciones no son capaces de dar una solución cerrada. Por ejemplo, cabe destacar que calcular la tasa de calor necesita un análisis de transferencia de radiación en la atmósfera, teniendo en cuanta la radiación de onda corta y onda larga, además de la transferencia de calor en procesos de evaporación, condensación y sublimación. Y además, para modelar el sistema en su totalidad también necesitamos modelar el océano y el hielo marino.

Océano

La mayor parte de ecuaciones que determinan la dinámica de los océanos están basadas en los mismos principios que las ecuaciones de la atmósfera, la única diferencia significativa es que la ecuación de la humedad específica no es necesaria y que hay que añadir una ecuación de salinidad. La ecuación de estado también es diferente.

Es más fácil calcular la tasa de calor en el océano que en la atmósfera, en este la única fuente de calor significativa es la radiación solar.

$$\frac{dT}{dt} = F_{sol} + F_{diff} \quad (7)$$

Donde F_{sol} es la absorción de radiación solar en el océano, F_{diff} hace referencia a la difusión. La ecuación no se aplica a la temperatura in situ, sino que se aplica a la temperatura potencial con el fin de considerar el efecto de la compresibilidad del agua del mar. Esta temperatura es la que tiene un elemento fluido que se mueve adiabáticamente a la superficie. Un elemento de agua si se mueve adiabáticamente (Q=0) de un punto de presión a otro con otra presión diferente experimentará una expansión o compresión debido a W, recordemos que ΔU = ΔQ - ΔW. Si una porción de agua se expande adiabáticamente su temperatura disminuye, y lo contrario sucede cuando se comprime adiabáticamente. En resumen, nos sirve para comparar la temperatura del agua con 2 puntos de presión diferentes. Cerca de la superficie la diferencia entre las 2 temperaturas es baja, sin embargo, en las capas profundas esta diferencia puede ser de hasta algunas decenas de grados.

La modelización del océano también tiene sus complejidades y es muy interesante, pero quizá no sea necesario ahondar mucho más en este aspecto para este trabajo de fin de máster. Los modelos también tienen en cuenta el hielo marino, la vegetación, los tipos de suelo, etc. Pero no podríamos explicarlos todos porque daría para más de un trabajo de fin de máster sólo esta parte.

Resolución numérica de las ecuaciones

La mayoría de las ecuaciones que dominan un sistema climático, a no ser que se simplifiquen mucho, son ecuaciones no lineales en derivadas parciales. Es necesario que el problema tenga una solución única que dependa de unas condiciones de contorno iniciales. Los modelos han de ser transformados a modelos numéricos para ser computados. Un método para resolver estas ecuaciones es el método de diferencias finitas, así al final tendremos una solución para cada uno de los puntos de la malla que se esté usando.

Imaginemos por ejemplo que tenemos que resolver la siguiente ecuación diferencial:

$$\frac{du}{dt} = A\cos(t) \quad (8)$$

La derivada respecto al tiempo se podría aproximar como una diferencia finita, dando lugar a la siguiente ecuación.

$$(U^{n+1} - U^n)\,/\Delta t = A\cos(n\Delta t) \quad (9)$$

Donde U_n es la solución discreta de la diferencia finita en el paso n. Si Δt es constante t=nΔt. Si sabemos Un entonces podemos resolver despejando para U_{n+1}. Estos problemas dependen fuertemente del valor inicial.

Para que un método numérico sea adecuado se tienen que cumplir dos propiedades fundamentales, en primer lugar Δt → 0 y Δx → 0, esto es esencial para garantizar que la ecuación se puede resolver. Si hacemos un desarrollo de Taylor:

$$U^{n+1} = U^n + \frac{du}{dt}\Delta t + \frac{1}{2}\frac{d^2u}{dt^2}\,\Delta t^2 + \cdots \quad (10)$$

El cual efectivamente tiende a du/dt cuando Δt tiende a 0. Lo que indica que es consistente.

La segunda propiedad que ha de cumplir es que la solución de la diferencia finita ha de converger a la solución de la PDE cuando x, t tienden a 0. Esta convergencia está relacionada con la estabilidad computacional, que dice que un esquema numérico es estable

computacionalmente si la solución de la ecuación de la diferencia finita permanece acotada si Δt tiende a 0.

Un criterio más general para determinar el paso de x y t mayores posible es el método de von Neumann, en el cual se puede expresar la ecuación de diferencia finita como un desarrollo en series de Fourier.

Discretización

Existen varias opciones para discretizar una ecuación. Además, para que sea consistente y estable para intervalos de tiempo razonablemente grandes el sistema ha de ser muy preciso y no ser demasiado complejo computacionalmente para no gastar demasiados recursos.

Tomemos, por ejemplo, una discretización por el método de Euler:

$$\frac{(U^{n+1}-U^n)}{\Delta t} = F(U^n) \tag{11}$$

Si quisiéramos resolver, por ejemplo, la siguiente ecuación, con k constante:

$$\frac{\partial u}{\partial t} = k\frac{\partial^2 u}{\partial x^2} \tag{12}$$

Podría ser discretizada como:

$$\frac{U_j^{n+1}-U_j^n}{\Delta t} = k\ \frac{U_{j+1}^n-2U_j^n+U_{j-1}^n}{\Delta x^2} \tag{13}$$

El índice j se refiere al punto número j de la malla espacial, que está a distancia (j-1)Δx . Este esquema es consistente si:

$$k\frac{\Delta t}{\Delta x^2} \leq \frac{1}{2} \tag{14}$$

La solución en j se actualiza a cada paso de tiempo n+1 desde los valores en tiempo n y j+1, j-1. Este proceso se repite iterativamente hasta tener todos los puntos iniciales.

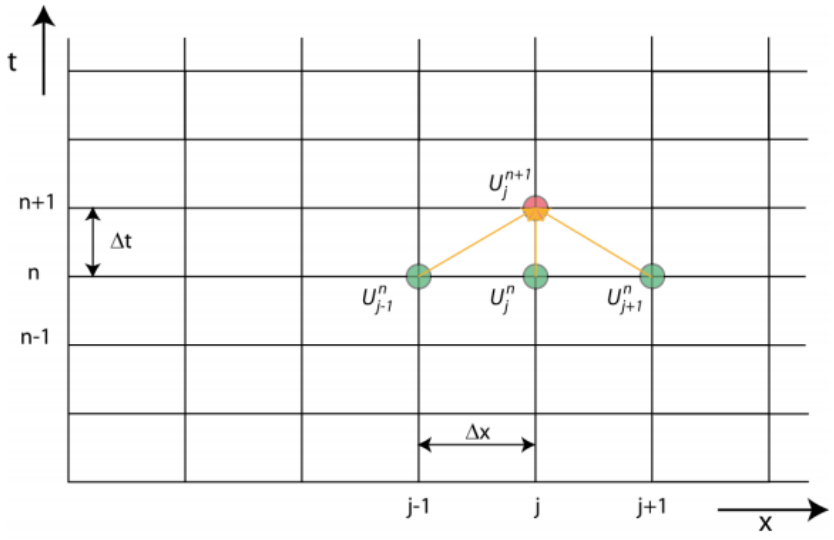


Figura 4: Esquema de la estructura de una red en espacio y tiempo con una sola dirección espacial mostrando las dependencias de U en las sucesivas iteraciones [7].

Finalmente volvemos a destacar la gran importancia que tiene el caos en estos procesos, ya que calculamos el siguiente paso a partir de las soluciones de los anteriores, por tanto, una variación, por muy ligera que sea de las condiciones de contorno puede hacer variar enormemente el estado final de la red. Este es uno de los mayores problemas a la hora de hacer previsiones meteorológicas.

Sección 1.2: Downscaling

Cada vez necesitamos pronósticos del tiempo con más resolución, imprescindibles en gran variedad de sectores socioeconómicos tales como la agricultura, el turismo, las aseguradoras, etc. Sin embargo, las predicciones de modelos de circulación global (GCMs) tienen una resolución limitada (cientos de kilómetros en aplicaciones climáticas). Para soslayar este problema existe el downscaling estadístico, que combina datos (predicciones) de los modelos GCM con datos históricos de estaciones locales con el fin de inferir relaciones estadísticas entre los GCMs de baja resolución y fenómenos de interés local; en la figura 5 se puede ver un esquema.

Predictores

Los predictores [3] son el input de los modelos de downscaling estadístico; normalmente son variables de gran escala que se usan para describir el estado en una región. En meteorología los más utilizados son la temperatura y la precipitación, pero también cabe destacar otros importantes como la altura geopotencial [4], medido en metros geopotenciales (la diferencia con los metros es que se considera la aceleración de la gravedad, que varía en función de la latitud y de la altura). Para la meteorología es mejor utilizar el geopotencial que la altura porque cuando el aire se desplaza sobre una superficie geométrica gana o pierde energía potencial, cosa que no pasa cuando se desplaza por una superficie geopotencial. Esta magnitud se calcula habitualmente para una presión en hPa dada, por ejemplo, si queremos la altura geopotencial a 500 hPa estamos midiendo la distancia desde la vertical sobre el mar hasta el punto de la atmósfera en el que la presión son 500 hPa, para este caso serían unos 5500 msn aproximadamente.

Formalmente se puede escribir como el geopotencial (trabajo requerido para elevar una unidad de masa una altura z sobre el nivel del mar) entre la gravedad inicial. Y las unidades son entonces m^2/s^2

$$Z = \frac{gz}{g_0} \tag{15}$$

Otro predictor que se puede utilizar es la humedad específica, esta indica la masa de vapor de agua en gramos en 1 kg de aire, conforme la temperatura desciende la cantidad de vapor de agua para alcanzar la saturación también desciende. Por tanto, si una masa de aire húmedo se enfría en aire ya no es capaz de mantener esa cantidad de agua y se produce una condensación que puede dar lugar a precipitación, por tanto, para predecir la precipitación un predictor útil es la humedad. Sus unidades son $\frac{g_{agua}}{kg_{aire}}$.

Se podría introducir más predictores, pero para este trabajo nos centraremos en la altura geopotencial, la temperatura y la humedad específica a diferentes presiones en la troposfera.

Predictando

El predictando es la variable dependiente (de salida) del modelo de downscaling estadístico, y típicamente está definida en una escala más pequeña que las variables predictoras. En nuestro caso consideramos la ocurrencia de precipitación en una serie de localidades.

Reanálisis

Los reanálisis [2] son productos generados con GCMs para caracterizar la configuración atmosférica en un período climático representativo (típicamente 30 años). Estos productos se generan asimilando las observaciones disponibles cada día en la rejilla del modelo, con el fin de sintetizar el estado del sistema global para un tiempo dado, como una instantánea del estado global de la atmósfera, desde la superficie hasta la estratosfera. Los reanálisis se pueden considera como las "observaciones" de la atmósfera caracterizadas en el espacio regular de la rejilla de un modelo. Estos datos se utilizan típicamente como "predicciones perfectas" en estudios de clima y están asociados con las ocurrencias meteorológicas día a día.

Downscaling estadístico

El clima local se puede definir como una combinación de la geografía local combinada con la circulación a gran escala. Por el momento los GCMs no son capaces de resolver este tipo de problemas debido a su insuficiente resolución espacial y al complejo sistema físico que subyace en las relaciones de pequeña y gran escala climática. Las técnicas de downscaling estadístico tratan de predecir el clima local (y) en base a las condiciones (predicciones) de larga escala dadas por los modelos (X), y a características locales/regionales (l):

$$y = f(X, l) \tag{16}$$

donde *f* representa al método estadístico particular utilizado. El downscaling estadístico consiste en un grupo heterogéneo de métodos que pueden variar en complejidad y aplicación. Todos ellos requieren una buena cantidad de datos observacionales de alta calidad, tanto reanálisis como series históricas de la variable local objetivo. Los métodos de downscaling estadístico [9] se pueden clasificar en 3 categorías: métodos lineales, clasificación de tiempo y generadores de tiempo. Para este trabajo se va a hacer downscaling empleando redes neuronales, que se explican en el siguiente capítulo, pero de todos modos vamos a recordar los métodos tradicionales.

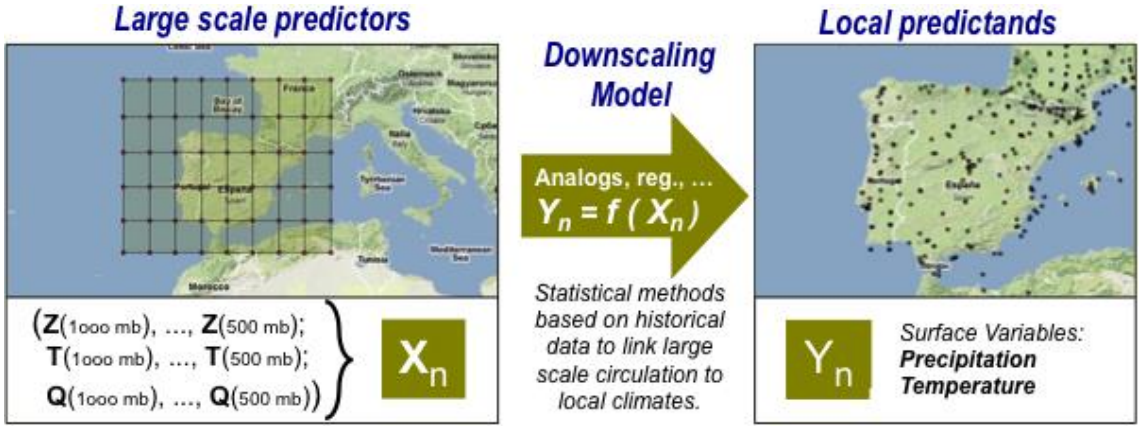


Figura 5: Esquema representativo del proceso de downscaling [8].

Métodos lineales

Establecen relaciones lineales entre los predictores y el predictando. Son unos métodos que se utilizan regularmente y la mayor restricción que presentan es la necesidad de una distribución normal de los valores de predictor y predictando, por ejemplo, para precipitación no serviría. Este tipo de métodos se utilizan para downscaling espacial principalmente, un ejemplo puede ser la regresión lineal.

Tipos de tiempo

La variable local se predice en base a "estados" de la atmósfera a gran escala. El estado futuro de la atmósfera, simulado en los GCM, se compara con su registro histórico. Ese valor corresponde a un valor o clase de valores de la variable local, la cual se reproduce con las condiciones de la atmósfera futuras. Estos modelos pueden usarse para hacer downscaling con funciones no normales, como la precipitación. Tienen el inconveniente de que es necesario disponer de un registro histórico muy grande, del orden de unos 30 años, con el fin de evaluar todas las condiciones posibles. Además, cabe destacar que son mucho más costosos que los métodos lineales. Un ejemplo de este tipo de método puede ser el método de análogos.

Generadores de tiempo

Son comúnmente usados en downscaling temporal, estos producen secuencias de valores diarios, pero como diferentes secuencias se pueden asociar a un set de datos, por ejemplo, valores mensuales, se suelen usar en modelos de impacto. Además, requieren largas secuencias de datos para implementarse y son bastante sensibles a datos erróneos o huecos.

Sección 1.3: Redes Neuronales

Las redes neuronales [10] junto a la paralelización suponen un cambio de paradigma en la computación, y cada vez más son más utilizados por parte de los científicos. La clave de esto son estructuras compuestas por un gran número de elementos interconectados o neuronas que trabajan en paralelo.

El rápido desarrollo de algunos campos de la inteligencia artificial, como el reconocimiento de imágenes, ha desarrollado soluciones a problemas con dificultades para hacer representaciones explícitas o razonamientos lógicos. Además del hecho de que los algoritmos estándar no funcionaban para resolver esos problemas. Las redes neuronales artificiales surgieron como una alternativa a la computación tradicional intentando emular las funciones de un cerebro humano. Este sistema se basa en que dados unos datos de entrada la neurona los computa y emite su propia señal. Esto se procesa por un mecanismo interno de la neurona, si el valor que emite alcanza un determinado umbral se emite un pulso eléctrico, y sino no.

Componentes de las Redes Neuronales

Neurona: Una neurona, o unidad de procesamiento, sobre un conjunto de nodos N, es un triplete (X, f, Y), donde X es un subconjunto de N, Y es un único nodo de N y f es una determinada función de activación que computa un valor de salida para Y basado en una combinación linear de nodos X.

Red neuronal: Una red neuronal artificial es un par (N, U), donde N es un conjunto de nodos y U es un conjunto de unidades de procesamiento sobre N que satisface que cada nodo de X perteneciente a N debe tener un input o un output para al menos una unidad de procesamiento en U.

Cada neurona ejecuta una simple operación con los inputs para obtener un output:

$$y_i = f(\sum_{j=1}^{n} w_{ij} x_j) \tag{17}$$

Donde f(x) es la función de la neurona con sus pesos w_{ij} que pueden ser negativos o positivos, reproduciendo el estado inhibido o excitado. Es de importancia el concepto de actividad lineal de la neurona, que es la suma ponderada de los pesos de otras neuronas.

$$Y_i = \sum_{j=1}^{n} w_{ij} x_j - \theta_i = \sum_{j=0}^{n} w_{ij} x_j \tag{18}$$

En algunos casos para tener en consideración el threshold del valor θ_i para la neurona y_i se conecta una nueva neurona auxiliar x_0=-1 a y_i con un peso $w_{i0} = \theta_i$. Ver figura 6.

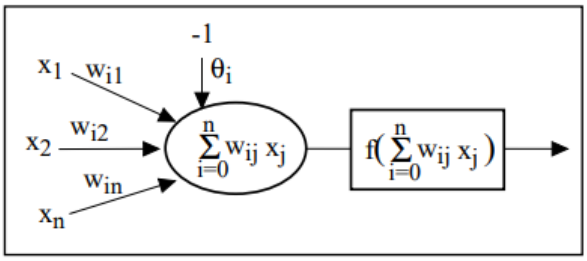


Figura 6: Esquema de una neurona consistente en una suma ponderada de los pesos de las neuronas de entrada, incluyendo el sesgo y la función [10].

Funciones de activación

La función de activación es la que se encarga de asignar si la neurona está activada o no dependiendo de si supera o no un cierto umbral. A continuación, se explican algunas de las más conocidas [11].

Función escalón

Estas funciones son muy simples, si el valor de y se encuentra por encima de un determinado threshold se considera activada, y si no pues no (ver figura 7).

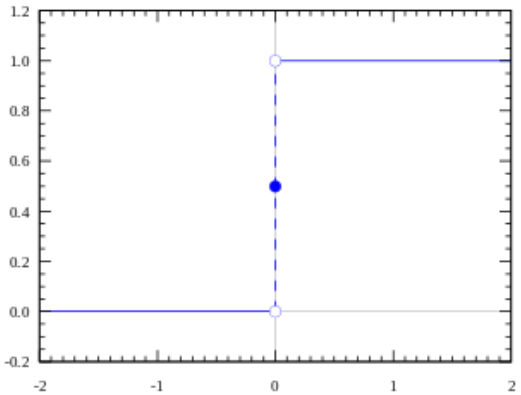


Figura 7: Representación de una función escalón [11].

Si queremos hacer un clasificador binario esta función podría servir, el problema viene con las clasificaciones multivariables, por ello no suelen ser utilizadas.

Función Lineal

Lo siguiente que se nos puede ocurrir para buscar esos estados intermedios es una función lineal, donde la activación es proporcional al input. El problema que presenta esta función es que su derivada es una constante. Por ejemplo, consideremos la recta A=cx, la derivada es c, es decir, el gradiente no tiene una relación con x. Además, para redes neuronales dificultaría el poner más de una capa, sería equivalente a tener una sola capa lineal combinación de las demás.

Sigmoide

$$A = \frac{1}{1+e^{-x}} \tag{19}$$

Esta función es suave, no es lineal, y sus combinaciones tampoco, por tanto, ahora sí que podemos poner capas con activación sigmoidal de manera sucesiva. Además, permite hacer bien una separación a partir de un cierto umbral dado, además la salida de la función de activación siempre va a estar en valores comprendidos entre 0 y 1 (ver figura 8).

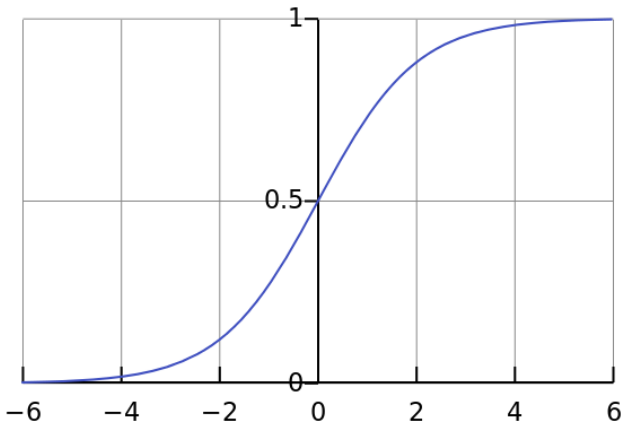


Figura 8: Representación de una función sigmoidal [11].

Como todo, tiene sus pros y sus contras, cuando estamos cerca de la parte cuasi-horizontal de la función cerca del 0 o 1, el gradiente es pequeño o desaparece (no supone ningún cambio porque es muy pequeño), lo que conlleva a que la red deje de aprender.

Una función muy parecida a la sigmoide es la tangente hiperbólica, es como una sigmoide, pero en lugar de tener el valor de la y acotado entre 0 y 1 lo tiene entre (-1,1) y también presenta el problema de que se puedan desvanecer los gradientes.

ReLU: Rectified Linear Unit

Esta función devuelve 0 si x es negativa y x si es positiva: A(x)=max (0, x) (ver figura 9).

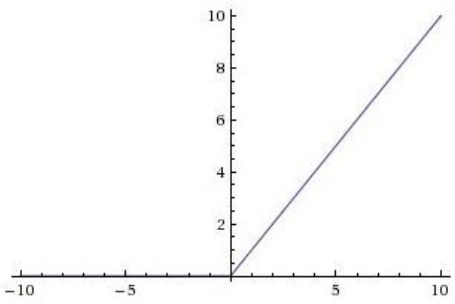


Figura 9: Función ReLU [11].

A simple vista puede parecer como una lineal, pero la ReLU no es lineal, y sus combinaciones tampoco, por lo que podemos poner sucesivas capas. Otro beneficio de las ReLU es que si la inicializamos con pesos aleatorios normalizados la mitad de la red tiene activación 0, por lo que se activan menos neuronas. Además, es menos costosa computacionalmente que la sigmoide y la tangente hiperbólica. Es una de las más usadas, sobre todo en problemas de deep learning dato que no tiene el problema de la sigmoide en los extremos que provoca esos desvanecimientos del gradiente.

Por poner algún punto negativo a estas funciones, la parte negativa de la función es 0, por tanto, sus gradientes también son 0. Eso quiere decir que esas neuronas que alcancen ese estado ya no van a responder a errores en el input. Este problema puede hacer que muchas neuronas dejen de responder, quedando pasiva buena parte de la red. Existen algunas funciones que son capaces de mitigar este efecto como las leaky ReLU, haciendo algo distinta de 0 la parte negativa, por ejemplo 0.01x. La idea sería impedir que el gradiente se haga 0 totalmente y que eventualmente se pudiera recuperar durante el entrenamiento.

Las redes neuronales artificiales tienen la capacidad de aprender de los datos. Una vez se elige la arquitectura de la red para un problema determinado, los pesos de las conexiones han de ser ajustados para captar la información de los datos en la estructura de la red. Después del proceso de aprendizaje es importante el poder comprobar la calidad del modelo resultante.

Existen múltiples formas de calcular medidas del error como, por ejemplo, la suma de los errores al cuadrado, RMSE, y el error máximo.

También cabe destacar la importancia de comprobar la calidad de nuestra predicción mediante algún proceso de cross-validación. Para ello los datos se dividen en 2 subconjuntos, una parte es de entrenamiento y otra de test. Cuando el error de test es mucho más grande que el de entrenamiento se dice que hay un problema de sobreajuste. Se sabe que cuando en estadística se usa un modelo con demasiados parámetros el modelo obtenido no es capaz de capturar el comportamiento real del mismo.

En un proceso de entrenamiento podemos distinguir 2 etapas, en una primera etapa el sesgo es alto, y tanto los errores de train como de test disminuyen conforme aumentamos el número de épocas. Llega un punto en el que el error de test ya no se reduce más mientras que el error de train sigue mejorando, es importante detener en este punto el entrenamiento porque de lo

contrario estamos sobre especializando el modelo y ya no es capaz de generalizar. Este procedimiento a veces se conoce también como early-stopping (ver figura 10), es decir, detener el entrenamiento cuando el error de test haya llegado a su mínimo, ya que se ahorra mucho tiempo computacionalmente hablando y además se evita el sobreajuste.

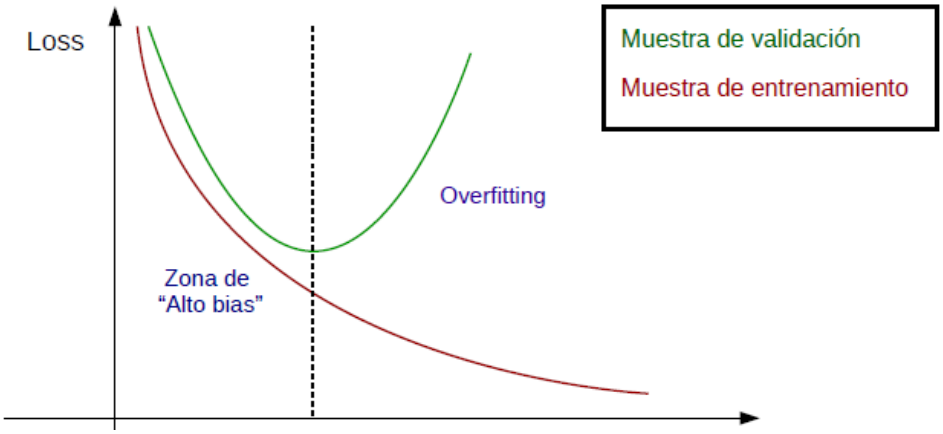


Figura: 10: Representación de la evolución de los errores de validación y train con el número de épocas, indicando la línea discontinua el early-stopping. (Apuntes de estadística)

Optimización para entrenar modelos de deep learning

Los algoritmos de optimización en modelos de deep learning se diferencian de los tradicionales en varios aspectos [12]. En machine learning medimos la calidad del algoritmo comparando con los resultados de test utilizando alguna función de coste, mientras que tradicionalmente solo nos centramos en el mínimo de la función de coste. Otra diferencia es que en deep learning usualmente no solemos parar en un mínimo local, en lugar de ello minimizamos alguna función de coste y se detiene el entrenamiento según algún criterio de convergencia que hayamos establecido basado en early stopping. Este early stopping se hace con la ayuda de una muestra de validación, en concreto cuando se alcanza un máximo de aciertos de la muestra de validación. Este hecho hace que muchas veces detengamos el entrenamiento, aun cuando las derivadas de la función de coste son altas. Al contrario que tradicionalmente donde este valor convergía cuando el valor de las derivadas era muy pequeño.

Batch y mini-batch

Los algoritmos de optimización que utilizan todo el conjunto de entrenamiento se llaman batch o deterministas, porque procesan todo el entrenamiento simultáneamente en un solo batch. Este término a veces se puede confundir ya que la palabra batch size se suele utilizar también entendiéndolo como mini-batch size. Los algoritmos que utilizan solo un ejemplo a la vez se denominan estocásticos. La mayoría de los algoritmos de deep learning están en un punto intermedio entre los deterministas y los estocásticos, utilizando más de un ejemplo, pero no todos. Estos son los métodos mini-batch, y ahora se los conoce también como estocásticos. Uno de los ejemplos más ampliamente utilizados es el descenso del gradiente, del que hablaremos un poco más tarde.

Entre las características de los tamaños de los mini-batch destaca que: batches grandes suelen proporcionar resultados algo más precisos, las arquitecturas multicore frecuentemente son infrautilizadas como batches muy pequeños. Esto motiva el utilizar un mínimo de tamaño de batch, ya que por debajo de algún determinado umbral no hay mejora ninguna.

Si todas las muestras en el batch han de procesarse en paralelo, la cantidad de memoria necesaria se escala con el tamaño del batch, lo que hace que en ocasiones el hardware pueda suponer un límite en el batch que queremos usar. Algunos tipos de hardware consiguen mejoras de tiempo con tamaños específicos de los arrays. Especialmente en GPU, donde se suelen utilizar potencias de 2 en los rangos de 32 a 256.

Descenso del gradiente

En modelos de deep learning consideramos la función objetivo como una suma finita de funciones [13]

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} f_i(x) \tag{20}$$

Donde $f_i(x)$ es la función de pérdida en cada iteración i. Cabe destacar que el coste computacional por iteración crece linealmente con el tamaño de n. El descenso del gradiente estocástico ofrece una solución más eficiente computacionalmente. En cada iteración, en lugar de calcular $\nabla f(x)$ se calcula muestras i, y se computa en su lugar $\nabla f_i(x)$. La cuestión es que el SGD utiliza $\nabla f_i(x)$ que es un estimador insesgado de $\nabla f(x)$ debido a que

$$\frac{1}{n}\sum_{i=1}^{n} \nabla f_i(x) = \nabla f(x) \tag{21}$$

En un caso generalizado, en cada iteración un mini-batch que consiste en índices de instancias de la muestra de entrenamiento, se puede escribir como

$$\nabla f_\beta(x) = \frac{1}{|\beta|}\sum_{i\epsilon\beta} \nabla f_i(x) \tag{22}$$

Y se actualiza x como

$$x = x - \eta \nabla f_\beta(x) \tag{23}$$

Donde β indica la cardinalidad del mini-batch y η indica la tasa de aprendizaje en cada paso.

Este es el mini-batch SGD. El coste computacional por iteración es del orden de β. Lo que hace que el coste computacional en cada iteración sea menor cuando el tamaño de mini-batch es más pequeño, no obstante tamaños de mini-batch mayores dan mejores resultados.

Redes Neuronales Convolucionales

Las redes neuronales convolucionales [14] o CNN, son un tipo especializado de redes neuronales que están pensada para trabajar con datos que se estructuren en una topología de malla, como es el caso de los datos de las variables predictoras de nuestro problema. Su nombre viene de las operaciones de convolución, su principal diferencia es que estas utilizan una convolución en lugar de una matriz convencional en al menos una de sus capas.

Convolución

Imaginemos, por ejemplo, que queremos determinar la posición de un objeto mediante tecnología láser, y dicho láser tiene algo de ruido. Para obtener una estimación con menos ruido tomaríamos la media de varias mediciones. Como es lógico, las mediciones más recientes tendrían más validez, por tanto, necesitaríamos que este hecho tenga más peso. Esto se puede hacer con una función que controle los pesos w(a), donde a es el tiempo que hace que se ha medido. Si aplicamos esa función a cada momento que sea medido tenemos que

$$s(t) = \int x(a)w(t-a)da \tag{24}$$

La operación de convolución a menudo se suele indicar con un asterisco:

$$s(t) = (x * w)t \tag{25}$$

En esta terminología, el primer argumento, x, se refiere al input, y el segundo, la w, hace referencia al kernel, el output se conoce también como mapa de características.

En aplicaciones de deep learning, el input normalmente es un array multidimensional de datos, y el kernel es otro array multidimensional de parámetros que se adaptan dependiendo del algoritmo. Estos arrays multidimensionales son tensores, de ahí el interés de utilizar TensorFlow para este tipo de problemas.

En las redes neuronales tradicionales cada unidad de output interacciona con cada input. Sin embargo, las redes neuronales convolucionales tienen interacciones dispersas (ver figura 11). Esto se consigue haciendo el kernel más pequeño que el input. Por ejemplo, cuando se procesa una imagen, el input puede tener miles de píxeles, pero somos capaces de detectar características importantes con kernels de solo decenas o centenas de píxeles. Lo que implica que necesitamos guardar menos parámetros y mejora la eficiencia del modelo requiriendo menos operaciones.

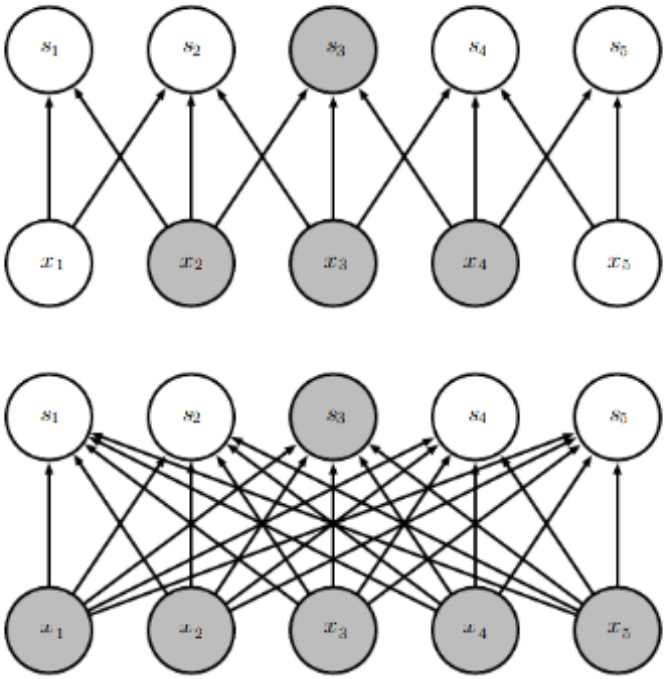


Figura 11: Conectividad dispersa, vista desde arriba. En gris se destaca s_3, y las neuronas x que afectan a s_3. Estas unidades son los campos receptores de s_3. Cuando s se forma con una convolución de kernel 3, solo 3 inputs afectan a s_3, mientras que cuando se hace con multiplicaciones de matrices es una red fully connected y todos los inputs afectan a s_3. [14]

Pooling

Una capa típica de una red neuronal convolucional consiste en tres fases diferentes. En una primera fase, la capa ejecuta varias convoluciones en paralelo para producir un conjunto de activaciones lineales. En la segunda etapa, cada activación lineal se ejecuta a través de activaciones no lineales, como las vistas anteriormente. En una tercera fase se puede utilizar una capa de pooling [15] con el fin de modificar el output.

Una función de pooling reemplaza el output de la red en una determinada posición por un aglomerado estadístico de los puntos colindantes. Por ejemplo, el max pooling devueleve el valor máximo de los vecinos próximos, existen muchas otras funciones de pooling, por ejemplo, la norma L2 o una media ponderada.

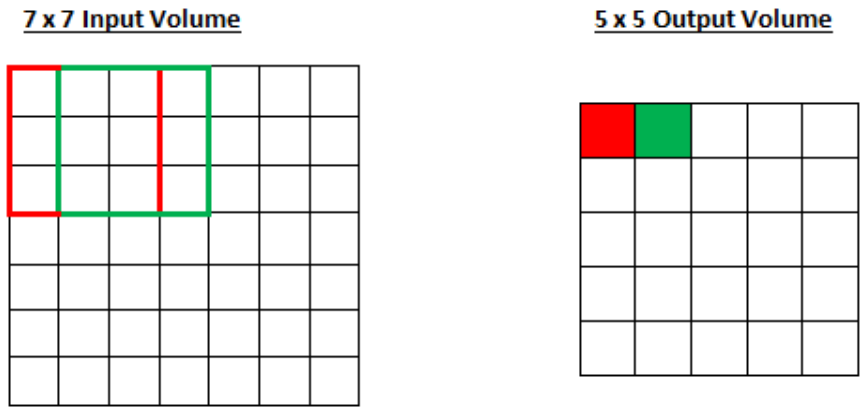
Stride y Padding

Cuando los filtros asociados a una red recorren el conjunto de datos nos encontramos con que en las esquinas principalmente no tenemos suficientes datos de los vecinos próximos, lo que hace que quizá no podamos estimar con tanta precisión las características de los bordes, por ejemplo, si se aplica un filtro de 3*3 en el centro de una capa de input de 8*8, para calcular el valor estaremos utilizando 9 casillas, sin embargo, si estamos en un extremo solo estaremos considerando 4 y no podríamos tener un output del mismo tamaño que el input. Para solventar este problema lo que se hace es expandir la red con 0 alrededor, por ejemplo, si se aplica un padding de 1 a un input de 8*8 se extiende a una malla de 10*10, una de 8*8 bordeado por una capa de 0, esto es el padding.

Otro factor a tener en cuenta es el stride, que hace referencia a cada cuantas casillas aplicamos los filtros, usualmente suele ser 1, es decir, se va recorriendo toda la malla de una casilla en una aplicando los filtros correspondientes, si quisiéramos un stride de 2, lo que implicaría sería que aplicaríamos los filtros pero esta vez tomando una casilla sí y la otra no, en la figura 12 podemos ver una representación de en qué consiste tanto el stride como el padding. La fórmula para calcular el tamaño del output de cualquier capa convolucional es:

$$O = \frac{W-K+2P}{S} + 1 \tag{26}$$

Donde O es el tamaño del output, W es el tamaño del input, K es el tamaño del filtro, P es el padding y S es el stride.



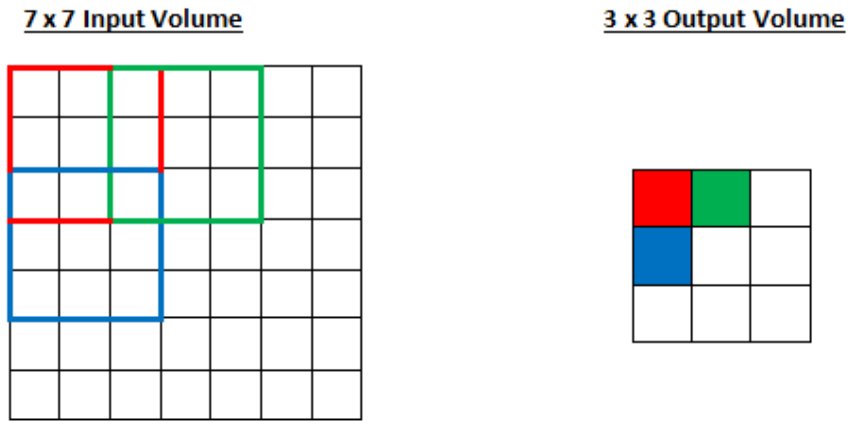


Figura 12: Arriba podemos ver claramente el tamaño del output tras aplicar filtros sin padding, abajo vemos el resultado de aplicar filtros con un stride de 2 [15].

Inicializaciones particulares de pesos

La red se puede inicializar con diferentes conjuntos de pesos aleatorios, en este trabajo se han utilizado las inicializaciones he_normal [16] y Xavier, ambas vienen preparadas en Keras dentro de TensorFlow. La inicialización he_normal devuelve muestras de una distribución normal centrada en 0 con una desviación estándar

$$\sigma = sqrt\left(\frac{2}{fan_{in}}\right) \tag{27}$$

Donde fan_{in} es el número de inputs en el peso del tensor.

Por otra parte tenemos la inicialización Xavier o Glorot_uniform, la cual devuelve muestras de una distribución uniforme en [-limit,limit], donde limit es

$$limit = sqrt(\frac{6}{fan_{in}+fan_{out}}) \tag{28}$$

Donde fan_{in} es el número de inputs en el tensor de pesos, y fan_{out} es el número de unidades de output en el tensor de pesos.

Curva ROC

Las predicciones probabilísticas pueden servir como ayuda para la toma de decisiones. Cada usuario de la predicción puede tener diferente sensibilidad a los fenómenos, por lo tanto, tendrá diferentes umbrales para establecer una alerta. Si un evento se predice y ocurre se considera un hit, y si se predice y no ocurre se considera una falsa alarma. La curva ROC [17] proporciona información de los hits y falsas alarmas en términos de probabilidad, en la figura 13 podemos ver distintos ejemplos.

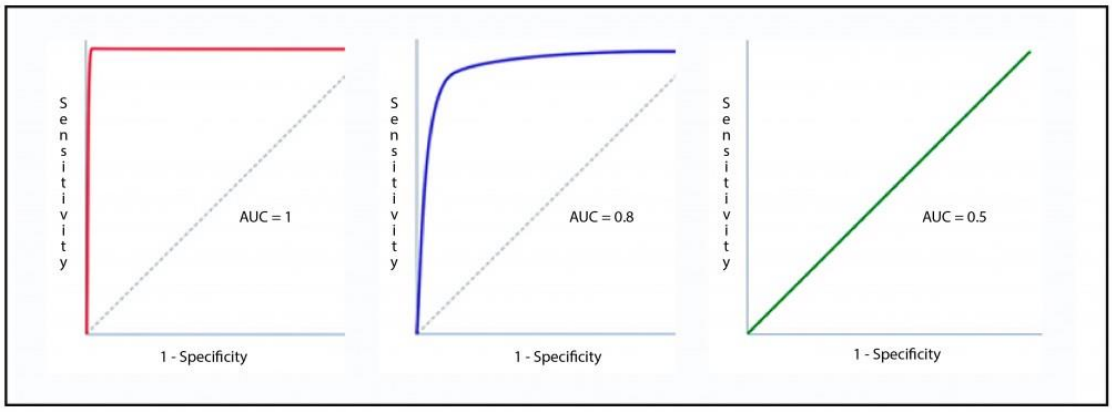


Figura 13: Diferentes curvas ROC, desde la perfecta, en rojo, hasta la aleatoria, verde. [18]

El ROC skill score es el área geométrica por debajo de la curva de ROC y proporciona una idea de la calidad de la predicción. En un pronóstico perfecto no hay falsas alarmas y todo lo que se predice es real, en este caso el área normalizada sería 1. En el caso de que no se tenga calidad de predicción se estiman aleatoriamente hits y falsas alarmas, por lo que el área en estos casos sería de 0.5.

En este trabajo se utiliza el ROC skill score como única medida de verificación de los resultados de downscaling estadístico. Una evaluación más sistemática está fuera de los objetivos de este trabajo.

Capítulo 2: Descripción del experimento

Experimento 1: Benchmark de VALUE (86 estaciones)

Como primer experimento de downscaling, se ha considerado el benchmark definido en el proyecto VALUE (intercomparación de técnicas de downscaling sobre Europa [22]). En este experimento se trata de predecir la precipitación local observada en 86 estaciones que cubren las principales zonas climáticas de Europa, considerando predictores "perfectos" dados por el reanálisis ERA-Interim. Los predictores utilizados han sido los siguientes: altura geopotencial (Z) a 1000, 850, y 500 hPa, la temperatura del aire(T) a 850 y 700 hPa, y la humedad específica (Q) a 850 y 700 hPa [19]. Estos datos se han interpolado a una rendija regular usando una interpolación de vecinos cercanos.

De la iniciativa VALUE tomamos los datos de precipitación que vienen dados de manera diaria en el periodo 1979-2008 para 86 estaciones distribuidas por Europa (ver figura 15). Para hacer el estudio se toma como muestra de entrenamiento el periodo (1985 – 2008) y de test (1979 - 1984). Sólo queremos predecir si llueve o no, así que establecemos un umbral de 1mm, los valores inferiores o iguales a esta cifra se considerarán días no lluviosos, mientras que los valores superiores se considerarán días lluviosos.

Con estos datos se utilizará una red neuronal (ver figura 14) para hacer el downscaling y se tomará como métrica el ROC skill score. Además, se hará un estudio comparativo CPU vs GPU para ver los tiempos de ejecución; por último, también se estudiará la correlación espacial y temporal de los resultados con las observaciones.

En todos los casos se ha entrenado el modelo un total de 30 épocas y una tasa de aprendizaje de 0.001 por estandarizar. En cuanto al tamaño de mini-batch se ha probado para las cantidades 32, 64, 128 y 256.

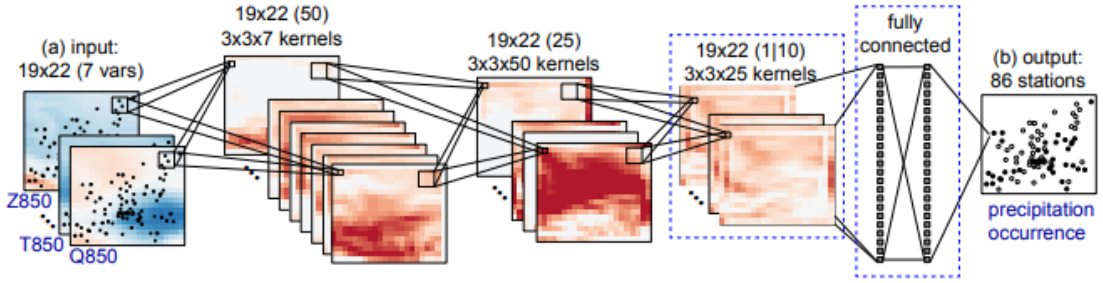


Figura 14: Esquema de la red neuronal con 86 estaciones [19]. La red pretende hacer un downscaling de la ocurrencia de precipitación en las 86 estaciones de VALUE tomando como base los 7 predictores de tamaño 19x22, para ello se crean 3 capas convolucionales con 50 kernels de tamaño 3x3x7, 25 kernels con tamaño 3x3x50 y otra de 1 kernel con tamaño 3x3x25. En todas las capas se ha utilizado una activación ReLU, excepto en la salida que se aplica una activación sigmoidal.

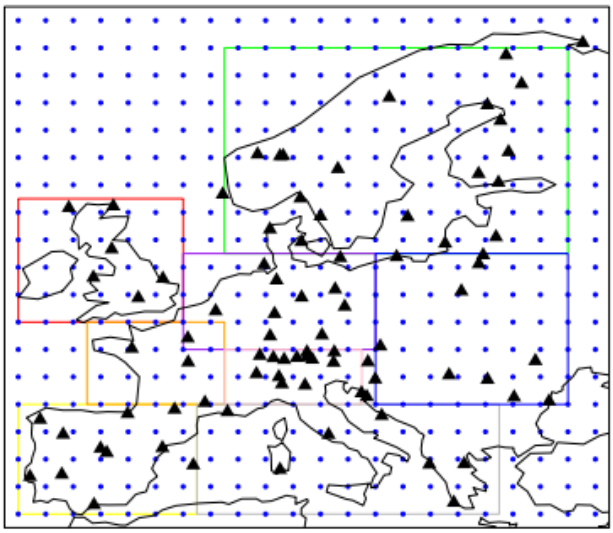


Figura 15: Representación de las 86 estaciones de VALUE sobre la rejilla de 2 grados de ERA-Interim [19].

Experimento 2: Escalado a toda Europa (3259 estaciones)

En este experimento se trata de analizar la escalabilidad del modelo utilizado en el experimento anterior, considerando un caso de estudio más realista, con una cobertura continental. Para realizar el downscaling hemos empleado exactamente los mismos predictores de ERA-Interim que en el caso anterior. La diferencia es que en este caso tomamos como predictando el valor de precipitación en una rejilla regular que cubre Europa con una resolución de ~ 50km. En particular se consideran los datos de E-OBS, una rejilla de observaciones interpoladas con 0.5 grados de resolución.

En VALUE los datos venían dados en estaciones, pero ahora en EOBS lo que tenemos son datos por cada cuadrícula del mapa. Por lo tanto, si cogemos el mismo dominio geográfico y temporal vemos que en lugar de 86 pasamos a tener un total de 3259 estaciones en tierra (ver figura 16). Para entrenar la red necesitaremos preprocesar los datos para que tengan la misma forma que las estaciones (ver anexos). Una vez preparados los datos repetimos el mismo análisis que en el caso anterior. Como apunte hay que señalar que en este caso también hemos probado el conjunto de test (1991-1996) y el resto como train, esto es así porque el conjunto inicial tenía 4483 de train y tan solo 324 de test, el nuevo conjunto tiene 3311 de train y 1496 de test.

Figura 16: Representación de las estaciones asociadas a las cuadrículas de tierra sobre Europa para EOBS en resolución de 0.5 grado [Hecha por Jorge Baño].

GPU

Las GPU o unidades gráficas de procesamiento se han utilizado en la industria de los videojuegos y el cine durante varias décadas [20]. A pesar de que sus aplicaciones iniciales consistían únicamente en hacer gráficos, estas han evolucionado hasta poder ser aplicadas para otros propósitos, como por ejemplo el análisis de imágenes.

Hasta hace poco, la potencia de las redes neuronales había estado restringida por las capacidades de las CPU. Las GPU permiten superar a las CPU haciendo que se reduzcan los tiempos en gran medida.

El acceso a GPU viene dado por diferentes interfaces como puede ser CUDA. Estas interfaces habilitan la paralelización de determinadas funciones específicas y permiten al software hacer los cómputos en los cores de GPU. La GPU que se utiliza en este TFM es una NVIDIA GeForce GTX 1080 Ti con 3584 cores.

Los cálculos de las redes neuronales son paralelizables, y se formulan como una sucesión de operaciones de matrices, un ejemplo de librerías que soportan estas operaciones son TensorFlow o Theano.

Espacio de trabajo

Trabajar con volúmenes de datos masivos requiere de equipos de altas prestaciones, pero normalmente un usuario habitual no tiene esos recursos disponibles en un ordenador doméstico, por ello, una herramienta imprescindible para la consecución de este trabajo de fin de máster son los servicios en la nube del IFCA. Estos servicios proporcionan a los usuarios ordenadores de altas prestaciones a los que se puede acceder remotamente y adecuarse a las necesidades, lo que ahorra una gran cantidad de espacio y dinero, ya que no es necesario tener las máquinas físicamente en nuestro centro de trabajo. También ahorramos dinero, ya que se ajusta a la cantidad de trabajo que tengamos y no hay que comprar equipos muy caros ni hacerles mantenimiento. En particular en este TFM el servicio se ha prestado de manera completamente gratuita, con un soporte técnico de calidad.

Respecto a las máquinas cloud usadas, en primer lugar, destacaré la máquina de datasciencehub que cuenta con 9 núcleos de CPU y hasta 128Mb de RAM, que la hace una máquina realmente buena para el preprocesado de las variables más pesadas. Es la misma

máquina que hemos utilizado los estudiantes del máster durante todo el curso para hacer nuestros trabajos. En segundo lugar, destacaré la utilidad de una segunda máquina, también facilitada por los servicios cloud del IFCA, que contiene 2 núcleos GPU NVIDIA con 11GB de RAM cada uno. En este trabajo se pretende introducir el uso de GPU para probar si realmente son útiles para disminuir los tiempos de ejecución en algoritmos con redes neuronales.

Para realizar los códigos se emplea el lenguaje de programación R junto con una serie de librerías diseñadas por parte del grupo de Meteorología y Minería de Datos del IFCA: loadeR, transformeR, downscaleR y visualizeR. Estas librerías están diseñadas expresamente para poder trabajar directamente con datos climáticos, lo que simplifica considerablemente el trabajo.

Para trabajar con las redes neuronales, tanto con GPU como con CPU, utilizamos la librería TensorFlow, originalmente diseñada para Python pero adaptada a R, TensorFlow es una poderosa herramienta orientada al machine learning de código abierto y creada por Google en 2015. Está centrada principalmente para resolver problemas numéricos y de redes neuronales mediante arrays multidimensionales o tensores.

Los datos se han obtenido de la página http://meteo.unican.es/udg-tap/home, que es el portal de acceso de datos del grupo de Meteorología de la Universidad de Cantabria, formado por personal del IFCA y del Departamento de Matemática Aplicada y Ciencias de la Computación. Desde este portal podemos acceder a los datos de E-OBS, ERA-Interim y muchos más, para ello sólo necesitamos crear un usuario y que nos autoricen. Una vez tenemos el acceso ya podemos trabajar con los datos directamente desde R de manera remota haciendo uso de las librerías del grupo citadas anteriormente. Cabe destacar la gran utilidad de este sistema ya que podemos acceder a todos los datos de manera remota sin tener que guardarlos en algún lugar de nuestro ordenador.

Capítulo 3: Resultados y Discusión

Resultados

En total hemos podido efectuar un total de 5 ejecuciones, tres de ellas asociadas al experimento 1 (86 estaciones) y 2 asociadas al experimento 2 (3259 estaciones). En la tabla 1 se representan los tiempos de ejecución obtenidos en todos los experimentos y en la figura 17 su representación gráfica. En el experimento 1 hemos alcanzado un ROC skill score de 0.76+-0.02, y en el experimento 2 de 0.79+-0.02.

Entorno|Batch	**32**	**64**	**128**	**256**
GPU86	4.0	3.7	3.6	2.7
CPU86	16.4	14.7	13.9	13.0
CPU3259_1	44.6	43.0	42.6	42.3
CPU3259_3	42.3	41.2	39.9	39.5
CPU86_variación	18.2	16.7	15.6	14.7

Tabla 1: Representación del tiempo en minutos de ejecución para cada prueba en función del tamaño de mini-batch. GPU3259_3 hace referencia a haber tomado el periodo de test 1991-1996. En el resto de casos _1 u omisión implica que se ha cogido el periodo de test 1979-1984. CPU86_variación es la prueba que se ha hecho con el código modificando la forma de obtener el ROC skill score con TensorFlow (anexo 5).

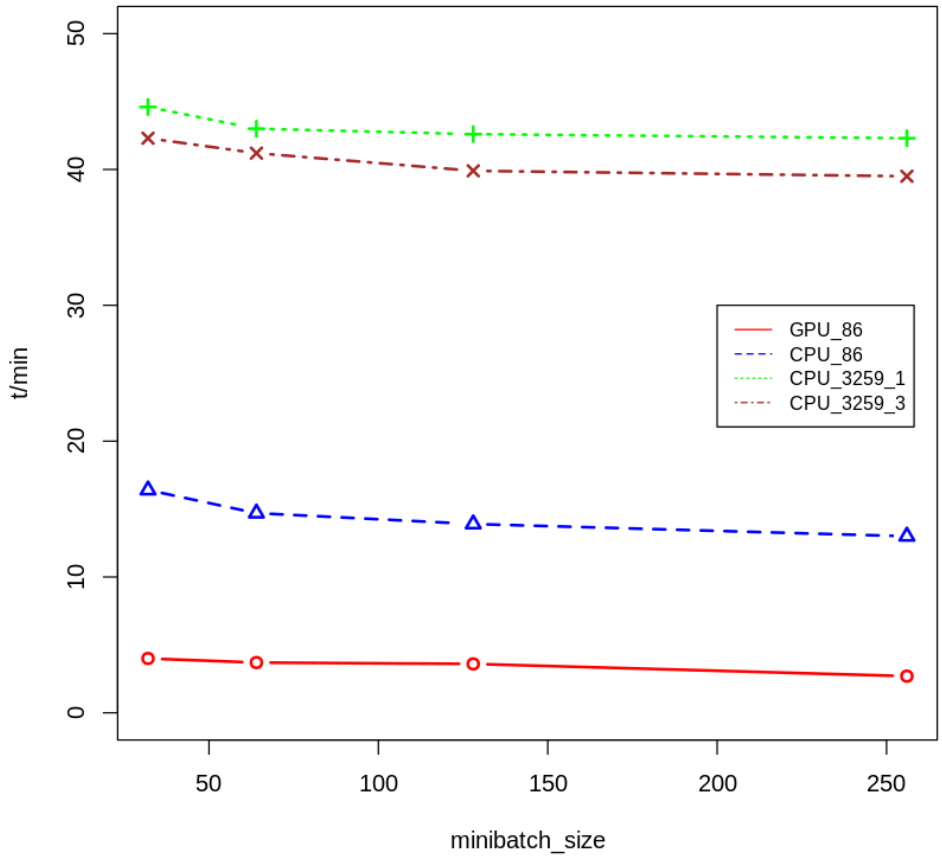


Figura 17: Representación del tiempo de ejecución del código en minutos respecto el tamaño de mini-batch indicando si es con 86 o 3259 estaciones. Los subíndices 1 y 3 indican el conjunto de train y test, el 1 hacer referencia al periodo de test 1979-1984 y el 3 al periodo de test 1991-1996.

Cabe destacar que no hay demasiada diferencia de resultados entre tomar las 86 estaciones de VALUE o las 3259 de EOBS, lo cual indica que se puede abordar el downscaling estadístico a escala continental de forma robusta y con buenos resultados. Además, el tiempo de ejecución para el Experimento 2 (EOBS) es sólo cuatro veces superior al del Experimento 1 (VALUE), a pesar de que el número de estaciones es casi 40 veces mayor; esto se debe a la estructura profunda de la red, donde sólo las capas finales están afectadas por el número de estaciones. Un factor importante a la hora de analizar los resultados es que en el caso de EOBS tenemos casi la mitad de los registros temporales que en VALUE, 4897/10958 en EOBS vs 9489/10958 en VALUE, lo que reduce los tiempos de ejecución a la mitad.

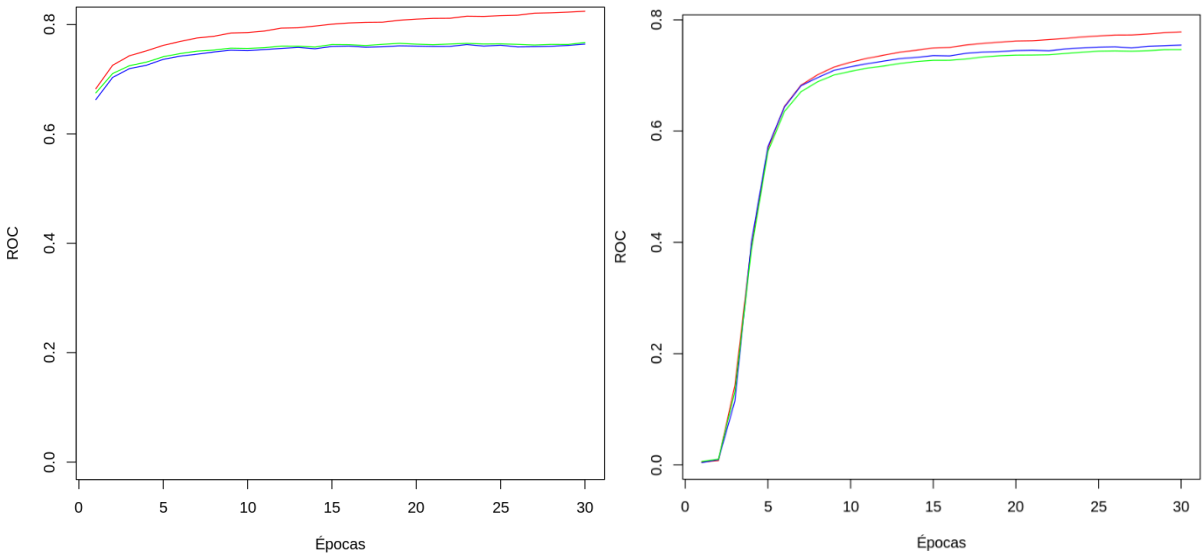


Figura 18: Ejemplo del entrenamiento de una red con mini-batch de 32 con 30 épocas a la izquierda, y con mini-batch de 256 a la derecha. En rojo el ROC skill de train, en azul el de validación y en verde el de test. Experimento 1.

En cuanto a los tiempos de ejecución se ha visto claramente que en el experimento 1 se ha conseguido reducir el tiempo de ejecución 4 veces si se usa GPU, lo que supone una mejora considerable. A pesar de haber sido capaces de probar el experimento 1 en GPU, en el caso del experimento 2 no hemos sido capaces de hacer la ejecución en GPU por problemas técnicos. No obstante, hemos tratado de hacer alguna modificación al código modificando la forma en la que calculábamos el ROC skill score, utilizando en este nuevo caso la librería TensorFlow en lugar de verification, con esto seríamos capaces de hacer la operación en GPU en lugar de CPU (ver anexo 5). Se ha probado esta modificación en CPU y los tiempos no se han reducido, sino que se han incrementado un poco. En teoría si se probara en GPU debería hacer el código algo más rápido.

En cuanto al tamaño del mini-batch se ha observado que cuanto más grande es menos ha tardado en ejecutarse, pero con diferencias mínimas de tiempo (ver figura 18). En el error de test no ha habido apenas diferencia, solo se ha visto en el error de train.

Otra cuestión que hemos analizado para ambos experimentos ha sido la correlación espacial y temporal. Para ello hemos calculado la correlación obtenida del modelo con la real. En las figuras 19 y 20 se pueden ver los resultados obtenidos.

Correlación temporal

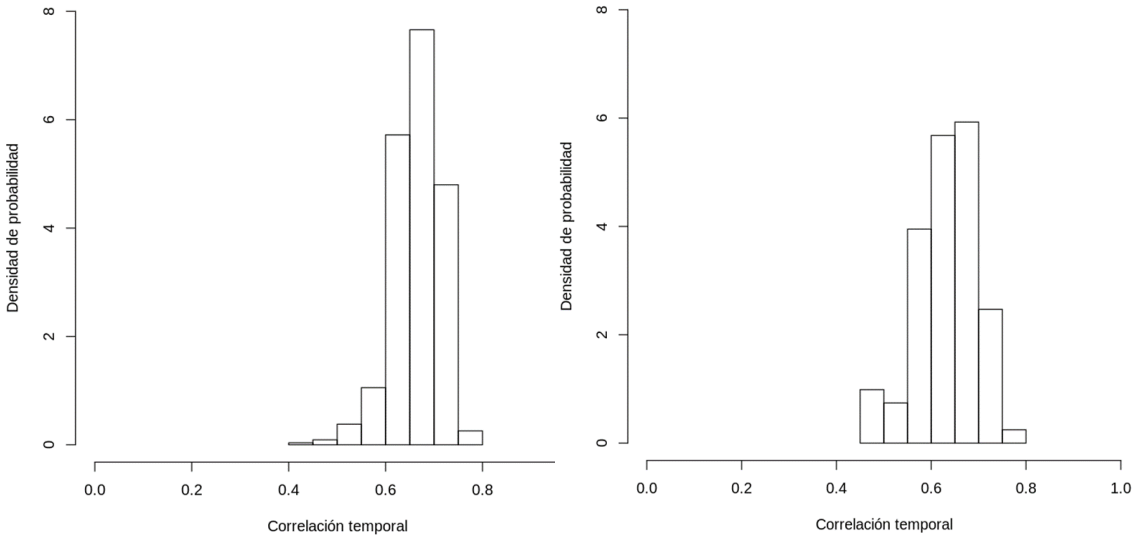


Figura 19: A la izquierda podemos ver el histograma de la densidad de probabilidad respecto la correlación temporal entre la muestra de test de entrada y la salida del modelo para 3259 estaciones y a la derecha lo mismo, pero para 86 estaciones.

Se puede ver como en el caso de la izquierda la función tiene algo menos de dispersión que en el de la derecha, lo que indica que la correlación temporal es más robusta en este caso. Esto se puede deber al hecho de tener un mayor número de estaciones, que ayudan a la red a predecir mejor cada localidad utilizando información de las vecinas (esta característica se denomina transfer learning).

Correlación espacial

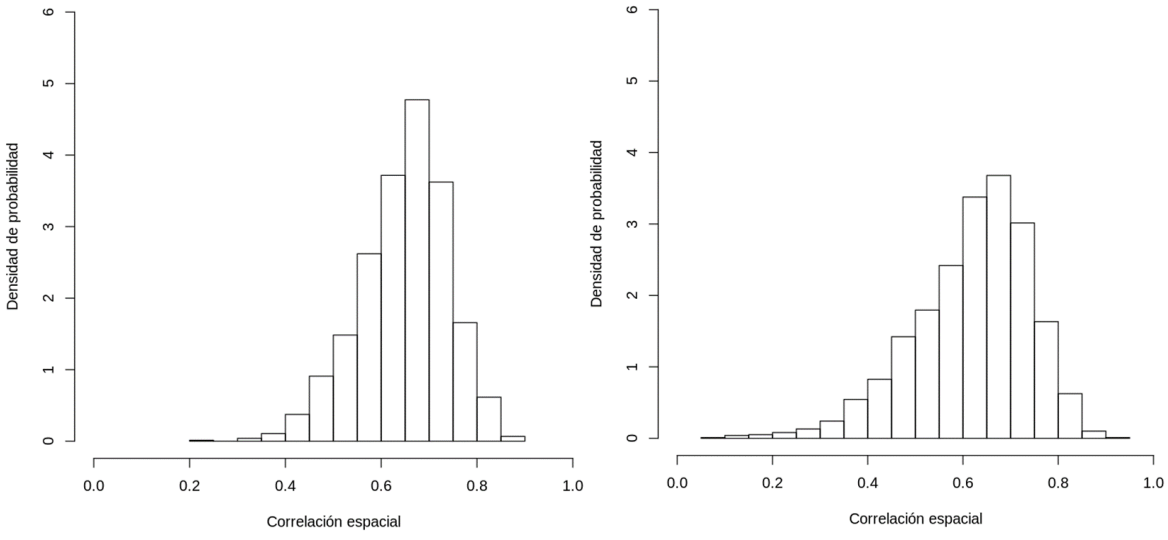


Figura 20: A la izquierda podemos ver el histograma de la densidad de probabilidad respecto la correlación espacial entre la muestra de test de entrada y la salida del modelo para 3259 estaciones y a la derecha lo mismo, pero para 86 estaciones.

En este caso, al igual que antes cuando tenemos 3259 estaciones vemos que la dispersión es ligeramente menor, lo que indica que en este caso el modelo predice mejor los patrones espaciales observados; no obstante, el comportamiento de ambas es más disperso, es decir, hay más dispersión en la correlación espacial (para distintos días) que en la temporal (para distintas estaciones). Este hecho indica que los patrones observados se predicen mejor para unos días que para otros, lo que da idea de la compleja relación entre la larga escala y la precipitación local (unos días existe mayor conexión entre estas escalas que otros).

Discusión

En cuanto al ROC skill score, que es la métrica que hemos utilizado para nuestra red podemos decir que los resultados han sido consistentes, ya que en el caso del experimento 1 con las 86 estaciones se ha podido ver como hemos obtenido un ROC skill score de 0.76+-0.02 considerando los casos estudiados, y en el de las 3259 estaciones 0.79+-0.02, errores inferiores al 5%.

En cuanto a los tiempos de ejecución se ha podido ver ligeras diferencias en el tiempo de ejecución dependiendo del número de mini-batch. Además, se ha probado a hacer un mismo caso en diferentes conjuntos de train y test y también ha habido una diferencia de unos 2 minutos en el tiempo de ejecución. Parece ser que cuando se aumenta el tamaño del mini-batch podemos llegar a entrenar más épocas antes de aplicar el early-stopping. El mini-batch pequeño da mejores resultados con menos épocas, pero también satura el entrenamiento antes.

En el caso de las correlaciones espaciales y temporales se han visto resultados comparables cuando considerábamos 86 estaciones que cuando considerábamos 3259 (con dispersiones ligermente inferiores en este último caso), lo que indica la escalabilidad de estos métodos en aplicaciones continentales. A diferencia de otros métodos de downscaling, donde los tiempos crecen linealmente con el número de estaciones, los modelos neuronales probados en este trabajo tienen un coste computacional similar cuando se aplican a muchas estaciones, sin que se empeoren los resultados. Esta es una buena propiedad de estos modelos.

Los resultados obtenidos con los datos de EOBS y VALUE son muy parecidos; sin embargo, para VALUE solo se han tomado 86 estaciones y para EOBS 3259. Esto indica que en VALUE se ha seleccionado cuidadosamente las estaciones sobre Europa para conseguir una representación óptima del continente.

Capítulo 4: Conclusión y Mejoras

Conclusión

Se ha podido comprobar que es posible hacer downscaling estadístico a escala continental con redes neuronales, se ha obtenido un ROC skill score que superaba el 0.75 en casi todos los casos, con diferencias mínimas dentro de las diferentes ejecuciones en cada experimento, lo que indica la adecuada robustez del método.

Otra de las conclusiones más importantes de este TFM es el hecho de que las GPU sí son capaces de disminuir los tiempos de ejecución. En este caso ha sido una reducción de 4 veces el tiempo en CPU, pero es probable que otros códigos o topologías de red den mejores resultados. Todo esto demuestra el gran potencial que tienen las GPU en problemas de deep learning.

Mejoras

Se podría continuar la investigación de este TFM intentando probar el código para GPU con 3259 estaciones y ver cuáles son los tiempos. Además, se ha visto que los resultados obtenidos con VALUE y con EOBS son muy parecidos, y el tiempo de ejecución con EOBS es del orden de 3-4 veces superior; por tanto, se podría probar el código con conjuntos de train y test provenientes de otros conjuntos de datos distintos y ver los resultados que se obtienen.

Otro campo por el que se podría ampliar la investigación es el probar diferentes arquitecturas de red neuronal con múltiples parámetros y capas diferentes para ver las diferencias en cuanto a tiempo y ROC skill score, este es un campo relativamente reciente, y seguro que es posible hacer alguna mejora sustancial.

También sería curioso probar algún algoritmo que incluya paralelización empleando varios núcleos de CPU o GPU, esto en teoría también disminuiría los tiempos de ejecución. Por ejemplo, se podría usar el gran potencial que tiene el supercomputador Altamira del IFCA para ver si merece la pena, o si por el contrario es suficiente con los recursos que tenemos.

Bibliografía

*Referencias revisadas por última vez el 30/08/2018

[1] https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/

[2] https://reanalyses.org/

[3] http://rcg.gvc.gu.se/edu/esd.pdf Rasmus E. Benestad , Deliang Chen & Inger Hanssen-Bauer, Norwegian Meteorological Institute, PO Box 43, 0313, Oslo, Norway, Earth Sciences Centre, Gothenburg University, Sweden June 15, 2007, Chapter 1 Introduction.

[4] http://int-dinamica-atmo.at.fcen.uba.ar/practicas/Apunte_Geopotencial.pdf

[5] https://www.meteorologiaenred.com/la-nina.html

[6]https://www.researchgate.net/figure/Model-of-the-two-modes-of-the-North-Atlantic-Oscillation-NAO-associated-storminess_fig32_319483260

[7] "Introduction to climate dynamics and climate modeling", Goosse H. et al, 2010, http://www.climate.be/textbook/pdf/Chapter_3.pdf

[8] https://meteo.unican.es/downscaling/intro.html

[9] A Review of downscaling methods for climate change projections USAID Septembre 2014. Trzaska S., Schnarr E, http://www.ciesin.org/documents/Downscaling_CLEARED_000.pdf

[10] Material de la asignatura Machine Learning I https://moodle.unican.es/pluginfile.php/389406/mod_resource/content/0/1999_BookCCGP_caps1_2_CORRECTED.pdf

[11] https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[12] "Deep Learning", Goodfellow et al, MIT Press, 2016, Chapter 8 http://www.deeplearningbook.org/contents/optimization.html

[13] https://gluon.mxnet.io/chapter06_optimization/gd-sgd-scratch.html

[14] "Deep Learning", Goodfellow et al, MIT Press, 2016, Chapter 9 http://www.deeplearningbook.org/contents/convnets.html

[15] https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

[16] https://keras.io/initializers/

[17] https://www.metoffice.gov.uk/research/climate/seasonal-to-decadal/gpc-outlooks/user-guide/interpret-roc

[18] https://www.cienciasinseso.com/en/tag/roc-curve/

[19] "Deep Neural Networks for Statistical Downscaling of Climate Change Projections", Baño-Medina J., Gutiérrez J.M., Herrera S. 2018.

[20] “Advancing drug Discovery via GPU-based deep learning”, Gawen et al. 2018, p579-p582, Expert Opinion on Drug Discovery. https://doi.org/10.1080/17460441.2018.1465407.

[21] “How machine learning could help to improve climate forecasts”, Jones N, 2018, Nature 548, p379-380, doi:10.1038/548379a.

[22] “An Intercomparison of a Large Ensemble of Statistical Downscaling Methods over Europe: Results from the VALUE Perfect Predictor Cross-Validation Experiment.” Gutiérrez, J. M., D. Maraun, M. Widmann, et al. 2018 .International Journal of Climatology. DOI: doi.org/10.1002/joc.5462.

Códigos en R

Anexo 1: Estandarización de datos ERA-Interim

```
#Creado por Jorge Baño

#Cargo librerías

library(tensorflow)

library(verification)

library(magrittr)

library(loadeR)

library(transformeR)

library(downscaleR)

library(visualizeR)

library(sp)

#Saco los predictores

loginUDG("*******", "*******")

ERA.url <- "http://meteo.unican.es/tds5/dodsC/interim/daily/interim20_daily.ncml"

dataInventory(ERA.url)

variables <- c("Z1000","Z850","Z500","Q850","Q700","T850","T700")

grid.list <- lapply(1:length(variables), function(x) {

  loadGridData(dataset = ERA.url,

               var = variables[x],

               lonLim = c(-10,32), # 22 puntos en total

               latLim = c(36,72),  # 19 puntos en total

               years = 1979:2008)

})

x <- makeMultiGrid(grid.list)

#Obtengo la y con los datos de VALUE

value <- tempfile(fileext = ".zip")
download.file("www.value-cost.eu/sites/default/files/VALUE_ECA_86_v2.zip",
              destfile = value)
y <- loadStationData(dataset = value, var = "precip", years = 1979:2008)
y <- binaryGrid(y,threshold = 1,partial = TRUE)

y <- binaryGrid(y, threshold = 1)

data <- dataSplit(x,y,
```

```
            f = list(
              c("1979","1980","1981","1982","1983","1984"),
              c("1985","1986","1987","1988","1989","1990"),
              c("1991","1992","1993","1994","1995","1996"),
              c("1997","1998","1999","2000","2001","2002"),
              c("2003","2004","2005","2006","2007","2008")),
            type = "chronological")
#Estandarizado de variables
xT <- data[[1]]$train$x ; yT <- data[[1]]$train$y
yT <- filterNA(yT); xT <- getTemporalIntersection(xT,yT,which.return = "obs")
xt <- data[[1]]$test$x  ; yt <- data[[1]]$test$y
xt <- scaleGrid(xt, base = xT, type = "standardize", spatial.frame = "gridbox")
xT <- scaleGrid(xT, base = xT, type = "standardize", spatial.frame = "gridbox")
xT <- xT$Data %>% drop() %>% aperm(c(2,3,4,1))
xt <- xt$Data %>% drop() %>% aperm(c(2,3,4,1))
yT <- yT$Data
```

Anexo 2: Entrenamiento red en GPU

```
#En este caso he modificado el código inicial de Jorge Baño para poder usar las GPU
#Cargo las librerias
library(verification)
library(magrittr)
library(sp)
#Preparo la GPU antes con el fin de que vaya aumentando el tamaño de la memoria utilizada en función de las necesidades
library(tensorflow)
config <- tf$ConfigProto()
config$gpu_options$allow_growth <- TRUE
#Todas las operaciones de la librería tensorflow se ejecutan en GPU
with(tf$device("/gpu:0"), {
  #Cargo los datos
  load("variables_est.RData")
  #Defino las variables de entrada
```

```
x  <- tf$placeholder(tf$float32, shape(NULL, 19L, 22L, 7L), name = "x")

y1 <- tf$placeholder(tf$float32, shape(NULL, 86L))

#Defino las capas de la red

h1 <- tf$layers$conv2d(x,filters = 50, kernel_size = c(3,3), padding = "same",
kernel_initializer = tf$contrib$keras$initializers$he_normal()) %>% tf$nn$relu(name = "h1")

h2 <- tf$layers$conv2d(h1,filters = 25, kernel_size = c(3,3), padding = "same",
kernel_initializer = tf$contrib$keras$initializers$he_normal()) %>% tf$nn$relu(name = "h2")

h3 <- tf$layers$conv2d(h2,filters = 1, kernel_size = c(3,3), padding = "same",
kernel_initializer = tf$contrib$keras$initializers$he_normal()) %>% tf$nn$relu(name = "h3")

h_flat <- tf$reshape(h3,c(-1L,418L), name = "h_flat")

y2 <- tf$layers$dense(h_flat,units = 86,activation = tf$nn$sigmoid, kernel_initializer =
tf$contrib$layers$xavier_initializer(uniform = FALSE))

#Función de pérdida

loss <- - tf$reduce_mean(y1*tf$log(y2) + (1 - y1)*tf$log(1 - y2), axis = 0L, keep_dims = TRUE,
name = "loss")

train_step <- tf$train$AdamOptimizer(learning_rate = 0.001)$minimize(loss)

init <- tf$global_variables_initializer()

#Inicio la sesion con la configuración establecida anteriormente

sess2 <- tf$Session(config=config)

sess2$run(init)

#Defino las épocas y el tamaño del mini-batch

num_epochs <- 30

size_batch <- 128

#Asigno un 10 % de la muestra de train a validación

indT <- sample(1:nrow(xT),size = round(0.9*nrow(xT)))

indV <- setdiff(1:nrow(xT),indT)

auc <- array(data = NA, dim = c(num_epochs,86,3))

#Mido el tiempo de entrenamiento

start<-Sys.time()

#Entreno la red

for (i in 1:num_epochs) {

  ind_batch <- indT[sample(1:length(indT),size = length(indT))]

  batch_x <- lapply(1:floor(length(ind_batch)/size_batch),function(z){
```

```
    xT[ind_batch[(size_batch*(z - 1) + 1):(size_batch*z)],,,]
  })
  batch_y <- lapply(1:floor(length(ind_batch)/size_batch),function(z){
    yT[ind_batch[(size_batch*(z - 1) + 1):(size_batch*z)],]
  })
  for (j in 1:length(batch_x)) {
    sess2$run(train_step,feed_dict = dict(x = batch_x[[j]], y1 = batch_y[[j]]))
  }
  pT <- sess2$run(y2,feed_dict = dict(x = xT[indT,,,]))
  pV <- sess2$run(y2,feed_dict = dict(x = xT[indV,,,]))
  pt <- sess2$run(y2,feed_dict = dict(x = xt))
  for (k in 1:86) {
    #Calculo el ROC skill score
    aucT <- roc.area(obs = yT[indT,k],pT[,k])$A*2 - 1
    aucV <- roc.area(obs = yT[indV,k],pV[,k])$A*2 - 1
    auct <- roc.area(obs = yt$Data[,k],pt[,k])$A*2 - 1
    auc[i,k,] <- c(aucT,aucV,auct)
  }
  #Visualizo los resultados
  print(c(mean(auc[i,,1]),mean(auc[i,,2]),mean(auc[i,,3])))
 }
 end<- Sys.time()
 print(end-start)
 sess2$close()
})
```

Anexo 3: Estandarización datos de E-OBS 0.5 grados

#En este código ha tenido una colaboración especial Maialen Iturbide en el filtrado del mar y el preparado de las fechas.

```
#Cargo las librerías
library(verification)
library(magrittr)
```

```
library(loadeR)
library(transformeR)
library(downscaleR)
library(visualizeR)
library(sp)
#Entro en UDG y cargo los predictores
loginUDG("******", "*******")
ERA.url <- "http://meteo.unican.es/tds5/dodsC/interim/daily/interim20_daily.ncml"
dataInventory(ERA.url)
variables <- c("Z1000","Z850","Z500","Q850","Q700","T850","T700")
grid.list <- lapply(1:length(variables), function(x) {
  loadGridData(dataset = ERA.url,
               var = variables[x],
               lonLim = c(-10,32), # 22 puntos en total
               latLim = c(36,72),  # 19 puntos en total
               years = 1979:2008)
})
x <- makeMultiGrid(grid.list)

UDG.datasets()$name[1:5]
#Predictando
eobs <-   loadGridData(dataset = "E-OBS_v14_0.50regular",
               var = "pr",
               lonLim = c(-10,32),
               latLim = c(36,72),
               years = 1979:2008)
y <- binaryGrid(eobs,threshold = 1,partial = TRUE)
y <- binaryGrid(y, threshold = 1)
data <- dataSplit(x,y,
                  f = list(
                    c("1979","1980","1981","1982","1983","1984"),
```

```
            c("1985","1986","1987","1988","1989","1990"),
            c("1991","1992","1993","1994","1995","1996"),
            c("1997","1998","1999","2000","2001","2002"),
            c("2003","2004","2005","2006","2007","2008")),
          type = "chronological")
#Tomo como muestra el conjunto 3
xT <- data[[3]]$train$x ; yT <- data[[3]]$train$y
#Sustituimos NA por -9999 para que el mar no moleste
ind.sea <- which(is.na(climatology(yT)$Data), arr.ind = T)[,-1]
for(i in 1:nrow(ind.sea)) yT$Data[, ind.sea[i,1] , ind.sea[i,2]] <- -9999
spatialPlot(climatology(yT))
#Se filtra con filterNA en las celdas de tierra:
yT <- filterNA(yT)
temporalPlot(yT)
#Se trata la hora de las fechas en yT para que la función getTemporalIntersection funcione
yT$Dates <- list("start" = gsub("00:00:00", "12:00:00", yT$Dates$start),
          "end" = gsub("00:00:00", "12:00:00", yT$Dates$end))
#Repetimos con los test
xt <- data[[3]]$test$x  ; yt <- data[[3]]$test$y
ind.sea2 <- which(is.na(climatology(yt)$Data), arr.ind = T)[,-1]
for(i in 1:nrow(ind.sea2)) yt$Data[, ind.sea2[i,1] , ind.sea2[i,2]] <- -9999
spatialPlot(climatology(yt))
yt <- filterNA(yt)
temporalPlot(yt)
yt$Dates <- list("start" = gsub("00:00:00", "12:00:00", yt$Dates$start),
          "end" = gsub("00:00:00", "12:00:00", yt$Dates$end))
xT <- getTemporalIntersection(xT,yT, which.return = "obs")
xt <- getTemporalIntersection(xt,yt,which.return="obs")
#Estandarizamos
xt <- scaleGrid(xt, base = xT, type = "standardize", spatial.frame = "gridbox", skip.season.check
= TRUE)
```

```
xT <- scaleGrid(xT, base = xT, type = "standardize", spatial.frame = "gridbox", skip.season.check = TRUE)

xT <- xT$Data %>% drop() %>% aperm(c(2,3,4,1))

xt <- xt$Data %>% drop() %>% aperm(c(2,3,4,1))

yT.coords <- expand.grid(yT$xyCoords$y, yT$xyCoords$x)[,2:1]

#Se ajusta la y para que tenga el mismo formato que en el caso de las estaciones

yTaux <- array3Dto2Dmat(yT$Data)

#Se eliminan las celdas de mar

yT <- yTaux[, which(yTaux[1,] != -9999)]

yT.coords <- yT.coords[which(yTaux[1,] != -9999), ]

yt.coords <- expand.grid(yt$xyCoords$y, yt$xyCoords$x)[,2:1]

ytaux <- array3Dto2Dmat(yt$Data)

#Se elimina el mar

yt <- ytaux[, which(ytaux[1,] != -9999)]

yt.coords <- yt.coords[which(ytaux[1,] != -9999), ]

#Guardamos los datos

save(xt,xT,yt,yT,file="varesteobs3.RData")
```

Anexo 4: Entrenamiento de la red EOBS-0.5

```
#Cargo las variables preprocesadas

  load("varesteobs3.RData")

  #Defino la entrada

  x  <- tf$placeholder(tf$float32, shape(NULL, 19L, 22L, 7L), name = "x")

  y1 <- tf$placeholder(tf$float32, shape(NULL, 3259L))

  #Defino las capas de la red

  h1 <- tf$layers$conv2d(x,filters = 50, kernel_size = c(3,3), padding = "same",
kernel_initializer = tf$contrib$keras$initializers$he_normal()) %>% tf$nn$relu(name = "h1")

  h2 <- tf$layers$conv2d(h1,filters = 25, kernel_size = c(3,3), padding = "same",
kernel_initializer = tf$contrib$keras$initializers$he_normal()) %>% tf$nn$relu(name = "h2")

  h3 <- tf$layers$conv2d(h2,filters = 1, kernel_size = c(3,3), padding = "same",
kernel_initializer = tf$contrib$keras$initializers$he_normal()) %>% tf$nn$relu(name = "h3")

  h_flat <- tf$reshape(h3,c(-1L,418L), name = "h_flat")

  #Salida
```

```
  y2 <- tf$layers$dense(h_flat,units = 3259,activation = tf$nn$sigmoid, kernel_initializer =
tf$contrib$layers$xavier_initializer(uniform = FALSE))

  #Función de pérdida

  loss <- - tf$reduce_mean(y1*tf$log(y2) + (1 - y1)*tf$log(1 - y2), axis = 0L, keep_dims = TRUE,
name = "loss")

  train_step <- tf$train$AdamOptimizer(learning_rate = 0.001)$minimize(loss)

  init <- tf$global_variables_initializer()


  sess=tf$Session()


  sess$run(init)

  #Establezco tamaño de mino-batch y número de épocas

  num_epochs <- 30

  size_batch <- 256


  #Tomo un 10% del train para validación

  indT <- sample(1:nrow(xT),size = round(0.9*nrow(xT)))

  indV <- setdiff(1:nrow(xT),indT)


  auc <- array(data = NA, dim = c(num_epochs,3259,3))


  #Mido el tiempo de entrenamiento

  start<-Sys.time()

  # Entreno la red

  for (i in 1:num_epochs) {

    ind_batch <- indT[sample(1:length(indT),size = length(indT))]

    batch_x <- lapply(1:floor(length(ind_batch)/size_batch),function(z){

      xT[ind_batch[(size_batch*(z - 1) + 1):(size_batch*z)],,,]

    })

    batch_y <- lapply(1:floor(length(ind_batch)/size_batch),function(z){

      yT[ind_batch[(size_batch*(z - 1) + 1):(size_batch*z)],]

    })
```

```
    for (j in 1:length(batch_x)) {
      sess$run(train_step,feed_dict = dict(x = batch_x[[j]], y1 = batch_y[[j]]))
    }
    pT <- sess$run(y2,feed_dict = dict(x = xT[indT,,,]))
    pV <- sess$run(y2,feed_dict = dict(x = xT[indV,,,]))
    pt <- sess$run(y2,feed_dict = dict(x = xt))
    for (k in 1:3259) {
      #Calculo el ROC skill score
      aucT <- roc.area(obs = yT[indT,k],pT[,k])$A*2 - 1
      aucV <- roc.area(obs = yT[indV,k],pV[,k])$A*2 - 1
      auct <- roc.area(obs = yt[,k],pt[,k])$A*2 - 1
      auc[i,k,] <- c(aucT,aucV,auct)
    }
    #Pinto el ROC de train, validación y test
    print(c(mean(auc[i,,1]),mean(auc[i,,2]),mean(auc[i,,3])))

  }
  end<- Sys.time()
  print(end-start)
  sess$close()
```

Anexo 5: Mejora propuesta código GPU

*No se ha podido implementar por problemas con la máquina virtual

Lo que se ha hecho es añadir un tensor para calcular el AUC y calcularlo específicamente con tensorflow en lugar de con librería verification, por lo cual en teoría se calcularía con las GPU y no las CPU. Se ha probado esta modificación en CPU pero no ha habido diferencias de tiempo.

```
x  <- tf$placeholder(tf$float32, shape(NULL, 19L, 22L, 7L), name = "x")
  y1 <- tf$placeholder(tf$float32, shape(NULL, 86L))
  h1 <- tf$layers$conv2d(x,filters = 50, kernel_size = c(3,3), padding = "same",
kernel_initializer = tf$contrib$keras$initializers$he_normal()) %>% tf$nn$relu(name = "h1")
  h2 <- tf$layers$conv2d(h1,filters = 25, kernel_size = c(3,3), padding = "same",
kernel_initializer = tf$contrib$keras$initializers$he_normal()) %>% tf$nn$relu(name = "h2")
```

```
h3 <- tf$layers$conv2d(h2,filters = 1, kernel_size = c(3,3), padding = "same",
kernel_initializer = tf$contrib$keras$initializers$he_normal()) %>% tf$nn$relu(name = "h3")

h_flat <- tf$reshape(h3,c(-1L,418L), name = "h_flat")

y2 <- tf$layers$dense(h_flat,units = 86,activation = tf$nn$sigmoid, kernel_initializer =
tf$contrib$layers$xavier_initializer(uniform = FALSE))

y3 <- tf$placeholder(tf$float32, shape(NULL))

y4 <- tf$placeholder(tf$float32, shape(NULL))


#Función de pérdida

loss <- - tf$reduce_mean(y1*tf$log(y2) + (1 - y1)*tf$log(1 - y2), axis = 0L, keep_dims = TRUE,
name = "loss")

train_step <- tf$train$AdamOptimizer(learning_rate = 0.001)$minimize(loss)


sess2 <- tf$Session()

#Defino el tensor del auc

roc<-tf$metrics$auc(y4,y3)

#Inicializo variables globales y locales

init = tf$group(tf$global_variables_initializer(), tf$local_variables_initializer())


sess2$run(init)

#sess$run(init)


num_epochs <- 30

size_batch <- 128


indT <- sample(1:nrow(xT),size = round(0.9*nrow(xT)))

indV <- setdiff(1:nrow(xT),indT)


auc <- array(data = NA, dim = c(num_epochs,86,3))


#Entreno la red

start<-Sys.time()
```

```
for (i in 1:num_epochs) {
 ind_batch <- indT[sample(1:length(indT),size = length(indT))]
 batch_x <- lapply(1:floor(length(ind_batch)/size_batch),function(z){
  xT[ind_batch[(size_batch*(z - 1) + 1):(size_batch*z)],,,]
 })
 batch_y <- lapply(1:floor(length(ind_batch)/size_batch),function(z){
  yT[ind_batch[(size_batch*(z - 1) + 1):(size_batch*z)],]
 })
 for (j in 1:length(batch_x)) {
  sess2$run(train_step,feed_dict = dict(x = batch_x[[j]], y1 = batch_y[[j]]))
 }
 pT <- sess2$run(y2,feed_dict = dict(x = xT[indT,,,]))
 pV <- sess2$run(y2,feed_dict = dict(x = xT[indV,,,]))
 pt <- sess2$run(y2,feed_dict = dict(x = xt))

 for(k in 1:86){
    #ROC skill socre directamente con tensorflow
   aucT <- sess2$run(roc,feed_dict = dict(y4 = yT[indT,k],y3 = pT[,k]))
   aucV <- sess2$run(roc,feed_dict = dict(y4 = yT[indV,k],y3 = pV[,k]))
   auct <- sess2$run(roc,feed_dict = dict(y4 = yt$Data[,k],y3 = pt[,k]))
   auc[i,k,] <- c(as.numeric(aucT[1]),as.numeric(aucV[1]),as.numeric(auct[1]))
 }
 #Pinto el ROC skill score
 print(c(mean(auc[i,,1])*2-1,mean(auc[i,,2])*2-1,mean(auc[i,,3])*2-1))
}
end<- Sys.time()
print(end-start)
sess2$close()
```