TRABAJO FIN DE GRADO

FUNCIONES DE PÉRDIDA DINÁMICAS EN MACHINE LEARNING: APLICACIÓN DEL ERROR CUADRÁTICO MEDIO DINÁMICO EN REDES NEURONALES ARTIFICIALES

TRABAJO FIN DE GRADO PARA LA OBTENCIÓN DEL TÍTULO DE **GRADUADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES**

SEPTIEMBRE **2021**

Eduardo Lavín Pallero

DIRECTOR DEL TRABAJO FIN DE GRADO:

Miguel Ruiz García

TUTOR DEL TRABAJO EN LA UPM:

Víctor Muñoz Villarragut

INDUSTRIALES
ETSII | UPM

POLITÉCNICA

"We can only see a short distance ahead, but we can see plenty there that needs to be done."

- Alan Mathison Turing.

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, Miguel Ruiz García, por dejarme continuar su trabajo de investigación y estar siempre dispuesto a ayudarme.

Me gustaría agradecer a mi familia el apoyo y el amor incondicional que me han mostrado, no solo durante la realización de este trabajo, sino a lo largo de toda mi vida.

También quiero agradecer a los amigos y compañeros que me han acompañado en los momentos dificiles de la carrera y han sabido sacarme una sonrisa cuando más lo he necesitado.

Por último, quiero dedicar este trabajo a mis padres, Cristina y Victor, por haber confiado siempre en mí y en mi potencial, y por haberme dado la oportunidad de recibir la formación como ingeniero de la que hoy puedo presumir.

Resumen ejecutivo

El aprendizaje automático (en inglés, *machine learning*) es una rama de la ciencia computacional y de la *inteligencia artificial* que se encarga de simular el proceso de aprendizaje en ordenadores. Esto se logra mediante el uso de algoritmos especializados, capaces de extraer propiedades intrínsecas de datos sin procesar. A partir de los patrones presentes en los datos, los algoritmos generan modelos de actuación que les permiten tomar decisiones aparentemente subjetivas.

En lo que se conoce como *aprendizaje automático supervisado* es necesario *entrenar* al algoritmo para que genere un modelo válido. Para ello, se muestra al algoritmo lo que se espera que obtenga al procesar lo que se conoce como unos *datos de entrenamiento*. Una vez *aprenda* estos datos de entrenamiento, se espera que este modelo sea capaz de procesar correctamente datos similares a los de entrenamiento.

Una de las principales aplicaciones del aprendizaje automático supervisado son las tareas de clasificación visual, ámbito en el que se desarrolla este Trabajo de Fin de Grado (TFG). Para una imagen como dato de entrada, el objetivo de la clasificación es asignarle correctamente la clase a la que pertenece dentro de un conjunto determinado de clases. Los algoritmos que resuelven estas tareas de clasificación visual presentan tres elementos fundamentales:

- Una *función de puntuación*, encargada de clasficar cada imagen asignado a cada una de las clases una puntuación.
- Una *función de pérdida*, que cuantifica la concordancia entre las predicciones y la clasificación real.
- Un *método de optimización*, mediante el cual se tratará de lograr que la clasificación del algoritmo sea lo más parecido a la realidad.

El proceso de aprendizaje consistirá entonces en un problema de optimización, en el que se minimizará el valor de la función de pérdida con respecto a los parámetros de la función de puntuación. La validez del modelo será medida mediante lo que se conoce como *precisión*, que muestra el porcentaje de imágenes correctamente clasificadas.

Los algoritmos que se emplean con más frecuencia en los problemas de clasificación de imágenes, como puede ser el reconocimiento facial o en la detección de objetos en la conducción autónoma, son los conocidos como *redes neuronales artificiales*. Consisten en un conjunto de unidades de procesamiento, dispuestas en un gran número de capas en serie interconectadas entre sí, las cuales trabajan en conjunto para aprender los patrones presentes en las imágenes asociados a cada categoría. En otras palabras, estos modelos pueden entenderse como una función no lineal, la cual depende de multiples parámetros que se optimizan para obtener el resultado deseado al introducir los datos de entrenamiento.

Uno de los grandes retos del aprendizaje automático en los últimos años ha sido mejorar el rendimiento de las redes neuronales. Estudios recientes se han centrado en el que seguramente es el proceso más importante dentro de los algoritmos de aprendizaje automático: la optimización de la función de pérdida.

Generalmente, las funciones de pérdida son funciones no-convexas: presentan una superficie con una gran cantidad de mínimos locales y puntos de silla, además de grandes variaciones en la curvatura. Si la red presenta pocos parámetros en comparación con la complejidad de la tarea a resolver (red subparametrizada), alcanzar el mínimo global de la función será muy complicado, si no imposible, en la mayoría de los casos. Por otro lado, si la red está altamente sobreparametrizada, alcanzar un mínimo global es relativamente sencillo y el esfuerzo se centra en la generalización del modelo: la precisión que obtiene en datos que no formaban parte de los datos de entrenamiento.

Normalmente existen mínimos locales lo suficientemente profundos como para que la red neuronal sea capaz de aprender. Aun así, sigue siendo un desafío alcanzarlos: la gran cantidad de inicializaciones, métodos de optimización y ajustes en la red hace que sea difícil encontrar la combinación adecuada de los mismos que permita descender adecuadamente por la hipersuperficie que la función de pérdida representa en el espacio de parámetros.

Para hacer frente a este problema, en *Tilting the playing field: Dynamical loss functions for machine learning* (Ruiz-García et al., 2021) se presentó la denominada *función de pérdida dinámica*. Lo novedoso de esta nueva función es que modifica su superficie *durante* el entrenamiento, permitiendo que se exploren nuevas regiones del espacio de parámetros en el proceso de aprendizaje. De esta forma, es posible alcanzar mínimos más profundos que logren mejores resultados para distintas configuraciones y combinaciones de los hiperparámetros de una red neuronal.

Para un conjunto de datos previamente dividido en clases, la función de pérdida dinámica introduce en una función de pérdida convencional unos factores dependientes del tiempo asociados a cada clase. Al entrenar la red, estos factores fomentan el aprendizaje de una de las clases sobre el resto durante un determinado periodo de tiempo y van oscilando de tal manera que se enfatiza una clase tras otra cíclicamente.

La función de pérdida dinámica fue inicialmente implementada en la conocida como *función de pérdida de entropía cruzada* (en inglés, cross-entropy loss). En redes que no logran encontrar mínimos profundos de la función de pérdida de entropía cruzada estándar, se demostró que la función de pérdida dinámica puede conducir a un incremento de la precisión del modelo. Incluso en redes que ya lograban buenos resultados de aprendizaje (redes sobreparametrizadas), las funciones de pérdida dinámica pueden conducir a una mejor generalización del modelo, entendiéndose como una mejora en el rendimiento ante nuevos datos de entrada.

El objetivo de este Trabajo de Fin de Grado es continuar esta línea de investigación, definiendo nuevas funciones de pérdida dinámicas usando como punto de partida el *error cuadrático medio* (en inglés, *mean squared error*), otra de las funciones de pérdida más populares en el aprendizaje automático.

A lo largo de este trabajo de investigación se realizará:

- *Una introducción a las redes neuronales para tareas de clasificación supervisada*, donde se explicará, tanto analítica como cualitativamente, el proceso de aprendizaje de las redes neuronales.

- *Un estudio detallado de la función de pérdida dinámica de entropía cruzada*, en el que mostrará la mejora en los resultados de aprendizaje que se obtienen al emplearla y se detallará el mecanismo de aprendizaje inherente. Se demostrará la presencia de un particular fenómeno presente durante el entrenamiento: aparece una inestabilidad del sistema que se manifiesta en las gráficas en forma de cascadas de bifurcaciones. La presencia de este fenómeno será explicado mediante los autovalores de la matriz Hessiana de la función y los autovalores del conocido como *Núcleo de la Tangente Neuronal* (en inglés, *Neural Tangent Kernel*).
- *La propuesta y el estudio de nuevas funciones de pérdida dinámicas usando el error cuadrático medio*. Se introducirán los factores dinámicos de dos formas diferentes en la función y se mostrará cómo en ambos casos también se produce una mejora en los resultados de aprendizaje. Se realizará de nuevo un estudio de la dinámica del aprendizaje de estas funciones, comparándolas con el caso de la entropía cruzada. Se observará de nuevo la presencia de inestabilidades durante el entrenamiento, empleando nuevamente los autovalores de la matriz Hessiana y del Núcleo de la Tangente Neuronal para explicar la aparición de estas inestabilidades.
- *La implementación de las diferentes funciones de pérdida dinámica en Python*, detallándose cada parte del código.

Palabras clave: aprendizaje automático, clasificación de imágenes, modelo, redes neuronales artificiales, función de pérdida dinámica, python.

Códigos UNESCO: 120601, 120707, 120711.

Índice general

1. Introducción

El aprendizaje automático (del inglés, *machine learning*) es una rama de la ciencia computacional que consiste en programar un ordenador de forma que sea capaz de resolver un problema basándose en la experiencia o en situaciones similares que tome como ejemplo. El aprendizaje automático es necesario para resolver problemas para los cuales el ser humano carece de la habilidad necesaria, o incluso en casos donde, a pesar de saber resolverlos, es complicado explicar la metodología a la que se recurre para ello. Por ejemplo, reconocer a alguien por su cara resulta una tarea fácil para una persona, se realiza de manera inconsciente; en cambio, si se quisiera explicar a un ordenador cómo hacer para reconocer una cara, sería un proceso que no resulta tan evidente. A pesar de ello, esta tecnología existe *¿Cómo es posible ?*

Para resolver un problema con un ordenador es necesario hacer uso de lo que se conoce como un *algoritmo*, una serie de instrucciones ordenadas capaces de llegar a la solución de un problema. De estos algoritmos se espera que generen un *modelo de actuación* frente a situaciones similares. En este sentido, el aprendizaje automático trabaja con algoritmos capaces de *aprender*, es decir, reconocer los *patrones* inherentes en la información que reciban de entrada. Para ello, es necesario emplear una gran cantidad de datos que permitan al algoritmo construir un modelo lo más preciso posible.

Volviendo al ejemplo del reconocimiento facial, si se procesa una gran cantidad caras de personas de diferente sexo, edad o raza, y se extraen las características propias que las definen, como puede ser la posición de los ojos o el ancho de la boca, se puede construir el algoritmo que sea capaz de diferenciar unas caras de otras y que genere un modelo que funcione con caras nuevas.

El aprendizaje automático está fuertemente ligado a la *inteligencia artifcial*, cuyo objetivo último es desarrollar sistemas que simulen la inteligencia humana. Para que un sistema sea inteligente, debe ser capaz de adaptarse a las condiciones ambientales cambiantes, por lo tanto, es necesario que aprenda. Si este sistema logra *aprender a aprender*, no es necesario que un programador se encargue de proponer soluciones a todas las situaciones a las que se puede enfrentar.

Generalmente, el aprendizaje automático se divide en tres grandes grupos en función de cómo sea la metodología de aprendizaje:

- *Aprendizaje supervisado.* Se muestra al algoritmo una serie de datos de ejemplo y lo que se espera que obtenga de ellos, de forma que el objetivo es encontrar el modelo capaz de lograrlo.
- *Aprendizaje no supervisado.* Se da al algoritmo unos datos de entrada pero sin mostrarle el resultado esperado. El algoritmo tiene entonces como objetivo encontrar por su cuenta los patrones en los datos de entrada.
- *Aprendizaje por refuerzo.* El algoritmo interactúa con un ambiente dinámico, en el cual debe realizar una tarea donde la única forma de conocer el resultado correcto depende de cómo reaccione ese ambiente.

El aprendizaje automático se puede aplicar en diferentes tareas, como puede ser el reconocimiento facial, la predicción de fenómenos meteorológicos, la detección de objetos o la conducción automática. A pesar de que existen numerosas aplicaciones, podemos agrupar estas tareas en:

- *Tareas de clasificación.* Consiste en asignar la clase a la que pertenece un dato de entrada con respecto a un determinado conjunto de clases. Por ejemplo, un algoritmo de clasificación capaz de separar el correo deseado del *spam*. Generalmente son tareas supervisadas.
- *Tareas de regresión.* Tiene como objetivo modelar la dependencia de la clasificación de unos datos con respecto a sus características para determinar cómo cambiará la clasificación según varíen las características. Por ejemplo, un algoritmo de regresión capaz de predecir el precio de una casa en función de cuantas habitaciones tenga. Generalmente son tareas supervisadas.
- *Tareas de agrupamiento.* Consiste en encontrar relaciones subjetivas en los datos y asociarlos en función de dichas relaciones. Un ejemplo puede ser la publicidad personalizada en función de las búsquedas en internet. Generalmente son tareas no supervisadas.

1.1. Objetivos y contenido del trabajo

Este Trabajo de Fin de Grado (TFG) se centra en una aplicación concreta del aprendizaje automático: las tareas de clasificación supervisadas. Dentro de este grupo de tareas, destacan las de clasificación visual, donde se emplean unos algorimos clasificadores que, generalmente, serán los conocidos como *redes neuronales artificiales*. Estos algoritmos emplean lo que se conoce como una *función de pérdida*, que permite cuantificar cómo de bien está clasificando su modelo las imágenes que toma como datos de entrada.

Existen diferentes tipos de función de pérdida que trabajan mejor o peor en función de la tarea que se les asigne. En un estudio realizado por el Dr. Miguel Ruiz García y su equipo de investigación en 2021, se presentó la denominada *función de pérdida dinámica* como una modificación de la ya existente *función de pérdida de entropía cruzada*.

Esta función dinámicapermite que la red neuronal artificial enfatice el aprendizaje de unas clases sobre otras durante un intervalo de tiempo. Por ejemplo, si se imagina un conjunto de puntos *rojos*, *amarillos* y *azules* que la red neuronal tuviese que clasificar visualmente, primero se centraría clasificar los *rojos* durante un tiempo, luego los *amarillos* y posteriormente los *azules*. Acto seguido, volvería a centrarse en los *rojos*, luego en los *amarillos* y finalmente los *azules*. Así hasta terminar el proceso de aprendizaje. En este estudio, se logró demostrar que este tipo de énfasis cíclico sobre cada una de las clases daba lugar a mejores resultados de aprendizaje en redes neuronales artificiales (Ruiz-García et al., 2021).

El objetivo de este Trabajo de Fin de Grado será continuar esta línea de investigación, trabajando en el desarrollo de una nueva función de pérdida dinámica, modificando la ya existente *función de pérdida de error cuadrático medio*. Por lo tanto que del contenido de la presente memoria se puede esperar:

- Una introducción a las tareas de clasificación de imágenes y a las redes neuronales artificiales.
- Un estudio detallado de la función de pérdida dinámica de entropía cruzada desarrolla por el Dr. Ruiz García y sus colaboradores.
- El desarrollo de una nueva función de pérdida dinámica empleando el error cuadrático medio y su implementación en una red neuronal.
- La presentación del código en lenguaje *Python* empleado para su aplicación.

- Una valoración del impacto de este estudio, una planificación temporal del trabajo y un cálculo del presupuesto de la investigación.
- Un resumen de las conclusiones a las que se ha llegado en este estudio.

2. Redes neuronales para el reconocimiento visual

Una de las principales aplicaciones del aprendizaje automático supervisado son las tareas de clasificación visual. Se trata de un concepto muy sencillo: dado un dato de entrada, generalmente una imagen, el objetivo es clasificarlo correctamente dentro de un conjunto determinado de clases.

Para que un algoritmo sea capaz de predecir la clase a la que pertenece una determinada imagen es necesario que *aprenda* de una serie de imágenes de referencia, a partir de las cuales generará un *modelo de clasificación*.

Por ejemplo, si queremos clasificar la imagen de un gato callejero dentro del conjunto de categorías [*gato*, *perro*, *barco*], inicialmente el algoritmo deberá aprender los rasgos característicos de cada clase. Para ello, tomará una batería de imágenes correctamente clasificadas de todas las categorías; posteriormente, *comparará* los rasgos de la imagen del gato con los que ha aprendido de cada una de las clases; finalmente, le asignará la categoría con la que presente mayor similitud.

Uno de las bases de datos usada con más frecuencia para comprobar el funcionamiento de este tipo de modelos es CIFAR10, el cual consta de 60000 imágenes en color de 32x32 píxeles, clasificadas en 10 clases. El conjunto prresenta de seis lotes de 10000 imágenes cada uno: cinco son lotes para entrenamiento y uno de ellos se emplea como evaluación, con 10000 imágenes aleatorias de todas las clases (Krizhevsky, 2009). La figura 2.1 muestra una pequeña parte de este conjunto de datos.

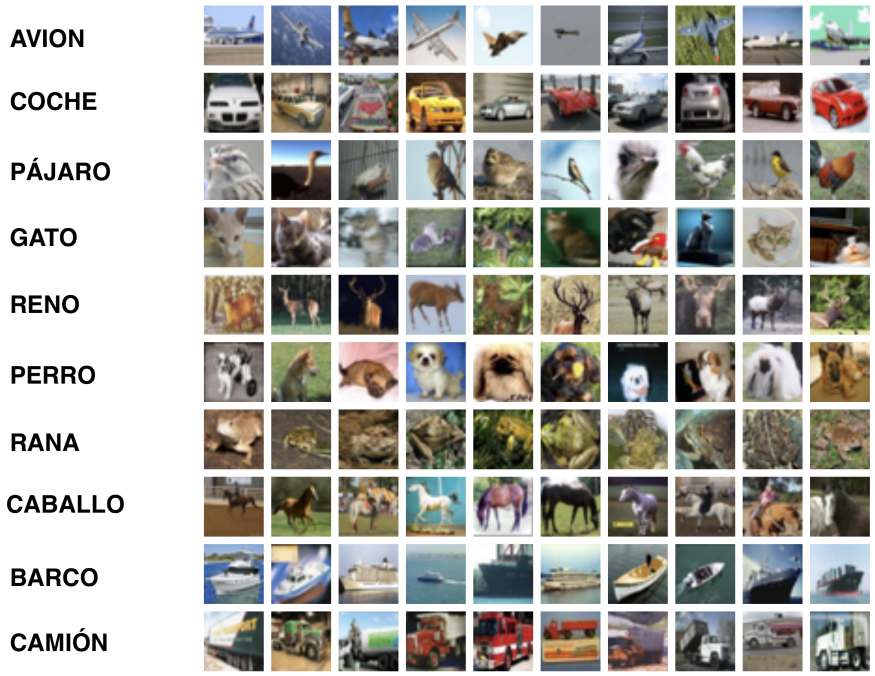

Figura 2.1: Muestra del conjunto de datos *CIFAR-10*. Imagen tomada de (Krizhevsky, 2009).

De manera general, la metodología que se sigue para abordar los problemas de clasificación puede resumirse en tres pasos (Karpathy, 2020):

1. Datos de entrada. Se tomará un conjunto de N datos, generalmente imágenes, cada uno de los cuales pertenece a una de las K clases en las que se puede categorizar. A estos se le conoce como *conjunto de datos de entrenamiento*.

2. Aprendizaje. El objetivo será entonces tomar el conjunto de datos de entrenamiento y aprender las características de cada una de las clases. A este proceso se le conoce como *entrenar al algoritmo* o *aprender un modelo de clasificación*.

3. Evaluación. Una vez aprendido el modelo, se evaluará la calidad del mismo empleándolo para predecir la clase a la que pertenece un nuevo conjunto de imágenes con las que el algoritmo no había trabajado hasta entonces. Posteriormente, se comparará la clasificación correcta de dichas imágenes con la predicha por el modelo.

En este capítulo se explicará el proceso de aprendizaje en los denominados *clasificadores lineales*, y de esta forma sentar una base para explicar los fundamentos de las *redes neuronales*. Más adelante, se hará una breve introducción a las *redes neuronales convolucionales* que incluyen operadores no lineales.

2.1. Clasificadores lineales

Tras definir el problema de clasificación de imágenes, es necesario introducir tres elementos fundamentales presentes en los algoritmos clasificadores: una *función de puntuación*, encargada de clasificar cada dato de entrada mediante una serie de puntuaciones asignadas a cada clase; una *función de pérdida*, que cuantifica la concordancia entre las predicciones y la clasificación real; y un *método de optimización*, mediante el cual se modificarán los parámetros de la función de puntuación para tratar de lograr que la clasificación del algoritmo sea lo más precisa posible.

El proceso de aprendizaje consistirá entonces en un problema de optimización, en el que se minimizará el valor de la función de pérdida con respecto a los parámetros de la función de puntuación (Karpathy, 2020).

2.1.1. Función de puntuación

Una imagen no es más que una matriz tridimensional de gran tamaño. Por ejemplo, una imagen de 32x32 píxeles, sabiendo que cada píxel presenta tres canales de color *RGB* (del inglés, *Red-Green-Blue*), básicamente se trata de un conjunto de 32x32x3 = 3072 números que van del 0 (negro) al 255 (blanco). Los píxeles de una imagen pueden disponerse en un vector de dimensión [Dx1], que en el caso del ejemplo, las dimensiones serían [3072x1] (Karpathy, 2020).

Se asume un conjunto N de imágenes $\boldsymbol{x}_i \in \mathbb{R}^D$ como datos de entrenamiento, cada una de las cuales está asociada a una clase $y_i \in [1...K]$, de manera que se tienen N imágenes de dimensión D y K categorías.

Se definirá ahora una función de puntuación tal que $f : \mathbb{R}^D \to \mathbb{R}^K$. Se trata de una función que va a asignar una puntuación por clase a cada imagen: la clase que presente la mayor puntuación será la que el clasificador asocie a la imagen. Una de las funciones más sencillas es la de asignación lineal:

$$f(\boldsymbol{x}_i, \boldsymbol{W}, \boldsymbol{b}) = \boldsymbol{W}\boldsymbol{x}_i + \boldsymbol{b} \tag{2.1}$$

En la ecuación 2.1, la matriz $\boldsymbol{W}$ contiene $[K\text{x}D]$ parámetros y es conocida como *matriz de pesos*. El vector $\boldsymbol{b}$, denominado *vector de sesgo*, contiene $[K\text{x}1]$ parámetros que influyen en la salida de la función pero sin llegar a interactuar con los datos $\boldsymbol{x}_i$.

La multiplicación matricial $\boldsymbol{W}\boldsymbol{x}_i$ evalúa K clasificadores simultáneamente, uno para cada clase, correspondiendo cada uno de ellos a una fila de $\boldsymbol{W}$. La salida de la función es un vector de puntuaciones $[K\text{x}1]$ resultante de sumar los vectores $\boldsymbol{W}\boldsymbol{x}_i$ y $\boldsymbol{b}$, de iguales dimensiones.

La figura 2.2 muestra un ejemplo sencillo de asignación de puntuaciones a una imagen, sin entrar en detalle sobre los valores de los parámetros. Al intentar clasificar la imagen de un gato dentro de las categorías [*gato*, *perro*, *barco*], el modelo ha dado la mayor puntuación a la categoría de *perro*, incluso ha asignado la menor puntuación a la categoría *gato*, indicando claramente que el modelo no ha sido entrenado correctamente.

Comúnmente se combinan todos los parámetros en una única matriz: el vector de sesgo se añade a la matriz como una columna a la derecha y se extiende el vector $\boldsymbol{x}_i$ añadiendo una componente al final, constante e igual a 1. Esto supone que la asignación de puntuaciones consiste en una única operación matricial. De ahora en adelante se denotará como $\boldsymbol{W}$ a la unión de todos los parámetros.

Los datos de entrada $(\boldsymbol{x}_i, y_i)$ son dados y fijos, pero los parámetros $\boldsymbol{W}$ pueden corregirse. La idea es modificar estos parámetros de manera que las puntuaciones asignadas coincidan con la clasificación real de las imágenes. Un vez finalizado el proceso de aprendizaje, las imágenes de entrenamiento pueden ser descartadas y los parámetros $\boldsymbol{W}$ finales son los que se emplearán para clasificar nuevas imágenes. Simplemente habrá que tomar las nuevas imágenes como entrada para la función de puntuación con estos parámetros.

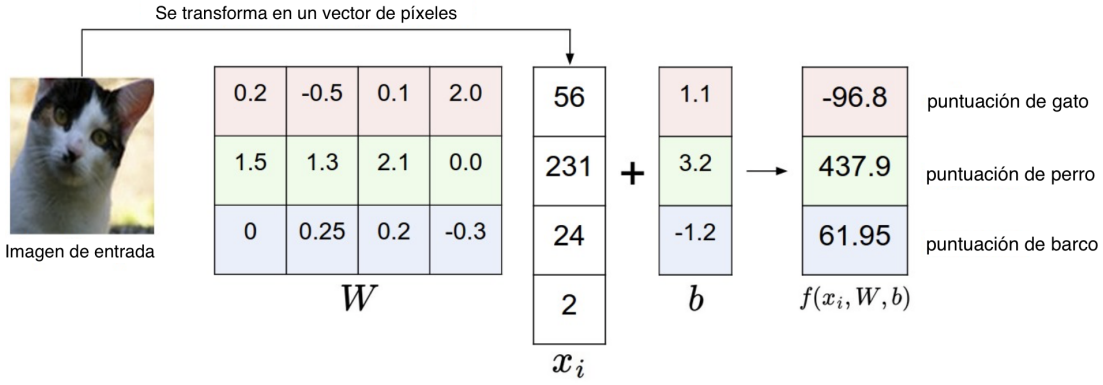

Figura 2.2: Ejemplo de asignación de puntuaciones a una imagen. Se asume que la imagen de entrada cuenta únicamente con 4 píxeles monocromáticos, para simplificar la representación. Se ha intentado clasificar la imagen dentro de las categorías [*gato* (rojo), *perro* (verde), *barco* (azul)], pero dadas las puntuaciones asignadas para cada clase, no ha habido éxito. Este conjunto de parámetros está convencido de que se trata de un perro. Imagen tomada de (Karpathy, 2020).

Se puede interpretar que cada fila de la matriz de pesos $\boldsymbol{w}_j$ (tras el entrenamiento) corresponde a un *prototipo* o *plantilla*, la cual recoge las características principales de la clase correspondiente según los datos de entrenamiento empleados. Al evaluar una imagen, mediante el producto escalar entre la fila y el vector de píxeles $\boldsymbol{x}_i$, conceptualmente se esta comparando este prototipo con la imagen, siendo la puntuación resultante mayor cuanta mayor similitud presenten.

Una imagen, expresada como un vector de píxeles $\boldsymbol{x}_i$, realmente es un punto en un espacio de dimensión D. El clasificador lineal dividirá este espacio en regiones correspondientes a las K distintas clases. Esta división viene definida por los hiperplanos $f(\boldsymbol{x}_i, \boldsymbol{W})_j = 0$, siendo $f(\boldsymbol{x}_i, \boldsymbol{W})_j$ la componente j del vector de puntuaciones. Al evaluar una imagen, la puntuación correspondiente a cada clase es la distancia del punto al hiperplano correspondiente. La optimización del modelo puede entenderse como la búsqueda de los hiperplanos que mejor separan los datos en el espacio de dimensión D.

En la figura 2.3 se muestra un ejemplo de clasificación lineal, en el que se quiere clasificar una serie de puntos de colores bidimensionales en dos clases distintas. La línea negra central representa el plano $f(\boldsymbol{x}_i, \boldsymbol{W})_j = 0$, siendo la puntuación asignada a cada punto la distancia con respecto al mismo. Concretamente, este clasificador es el conocido como *Máquina de Soporte Vectorial Multiclase*, el cual será presentado más adelante, que separa el espacio de tal forma que exista un margen entre los puntos y el plano, representado en líneas discontinuas.

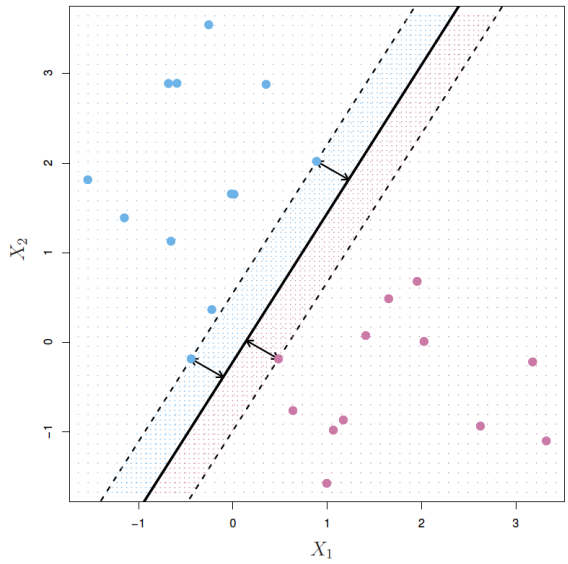

Figura 2.3: Ejemplo de clasificación lineal en el caso de tener dos clases bidimensionales. La linea negra central representa el plano $f(\boldsymbol{x}_i, \boldsymbol{W})_j = 0$. Imagen tomada de (Tandel, 2017).

2.1.2. Función de pérdida

Una vez definida la función de puntuación, es necesario definir una función de pérdida que permita cuantificar cómo de bien está trabajando el modelo con los parámetros actuales. La *pérdida* será mayor cuanto peor se esté clasificando los datos de entrada y será cero si todos los datos están correctamente clasificados.

La función de pérdida va a determinar el rendimiento de un algoritmo de clasificación. Dos de las principales funciones de pérdida empleadas en las tareas de clasificación con modelos lineales, son la *función de pérdida de bisagra* y la *función de pérdida de entropía cruzada.*

Recuérdese que se está trabajando con un vector de píxeles de una imagen $\boldsymbol{x}_i$, el cual tiene una clase y_i asociada, que se evalúa con una función de puntuación lineal $f(\boldsymbol{x}_i, \boldsymbol{W})$ igual a la presentada en la ecuación (2.1), pero en la que se han unificado todos los parámetros en $\boldsymbol{W}$.

- Función de pérdida de bisagra

Esta función de pérdida es comunmente empleada en un tipo de algoritmo conocido como *Máquina de Soporte Vectorial Multiclase* (en inglés, *Multiclass Support Vector Machine*. Esta función toma la forma

$$\mathcal{L}_i = \sum_{j \neq y_i} max(0, \boldsymbol{s}_j - \boldsymbol{s}_{y_i} + \Delta), \tag{2.2}$$

siendo $\boldsymbol{s}_i = f(\boldsymbol{x}_i, \boldsymbol{W})_j$ la puntuación para la clase j.

Para una imagen representada por $\boldsymbol{x}_i$, se trata de una comparación entre la puntuación de la clase correcta y_i y la puntuación del resto de clases. La función de pérdida fuerza a que la puntuación de la clase correcta sea mayor que la puntuación de las clases incorrectas en, al menos, una cantidad Δ. Cuando $\boldsymbol{s}_i - \boldsymbol{s}_{y_i} < \Delta$, la salida de la función es cero, lo cual implica que se considera que el modelo ha clasificado correctamente la imagen. En caso contrario, $\mathcal{L}_i > 0$, por lo que se considera que el modelo ha fallado.

- Función de pérdida de entropía cruzada

En este caso, el clasificador *Softmax* emplea la llamada *función de pérdida de Entropía Cruzada*, que se puede definir como

$$\mathcal{L}_i = -\log(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}) \qquad \text{,o equivalentemente,} \qquad \mathcal{L}_i = -f_{y_i} + \log(\sum_j e^{f_j}), \tag{2.3}$$

donde se denota como f_j la componente j del vector de puntuaciones $f(\boldsymbol{x}_i, \boldsymbol{W})_j$, de forma que f_{y_i} es la componente correspondiente a la clase correcta de la imagen.

La función $g_i(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_j e^{z_j}}$ es conocida como la *función Softmax*. Al emplear esta función en todas las componentes de un vector real $\boldsymbol{z}$, éste se transforma en un vector normalizado: todas sus componentes toman valores en el intervalo $[0, 1]$ y la suma de todas es siempre igual a 1.

Cuando se utiliza esta función de la forma $g_i(f) = \frac{e^{f_{y_i}}}{\sum_j e^{f_j}}$, se está calculando la *probabilidad* de que la clase y_i sea la asignada a la imagen $\boldsymbol{x}_i$ dados los parámetros $\boldsymbol{W}$. Al tomar esta función como argumento del logaritmo en la función de entropía cruzada de la ecuación 2.3, si la puntuación asignada a la clase correcta con respecto al resto de clases es muy elevada, entonces $g_i(f) \sim 1$, por lo que la función de perdida $\mathcal{L}_i \sim 0$. Cuanto menor sea la puntuación asignada a la clase correcta, es decir, a medida que $g_i(f)$ toma valores cada vez más cercanos a 0, mayor será el valor de la pérdida $\mathcal{L}_i$.

- *Penalización de regularización*

Una vez se ha calculado $\mathcal{L}_i$, ya sea mediante la función de pérdida de bisagra o la de entropía cruzada, es posible definir un nuevo término para calcular la pérdida total del modelo: la *penalización de regularización*.

Un conjunto de parámetros $\boldsymbol{W}$ que clasifican correctamente cada uno de los datos de entrenamiento, puede no ser único. Por ejemplo, cualquier múltiplo de los mismos de la forma $\lambda \boldsymbol{W}$, siendo $\lambda > 1$, también va a clasificar correctamente los datos según cualquiera de las dos funciones de pérdida presentadas.

Para eliminar esta ambigüedad, se emplea la penalización de regularización. Una de las más empleadas en las tareas de clasificación es la norma cuadrática $L2$, que permite descartar los pesos de gran tamaño, y viene dada por la expresión

$$R(\boldsymbol{W}) = \sum_k \sum_l {w_{k,l}}^2. \tag{2.4}$$

En la ecuación (2.4) se denota cada elemento de la matriz de pesos $\boldsymbol{W}$ como $w_{k,l}$. Nótese que la penalización no toma en cuenta los datos de entrada $\boldsymbol{x}_i$.

Dados N datos de entrenamiento y un hiperparámetro $\lambda > 1$ conocido como *fuerza de regularización*, podemos definir la pérdida total del modelo en función de dos componentes: la primera será la pérdida media $\mathcal{L}_i$ de todos los datos de entrenamiento, el cual se denominará *pérdida de datos*, y la segunda será la penalización de regularización multiplicada por λ, siendo ésta la *pérdida de regularización*. La expresión de la pérdida total toma la forma

$$\mathcal{L} = \frac{1}{N} \sum_i \mathcal{L}_i + \lambda R(\boldsymbol{W}). \tag{2.5}$$

2.1.3. Método de optimización

Ya se ha explicado cómo la función de puntuación, tomando un conjunto de parámetros $\boldsymbol{W}$, es capaz de predecir la clase a la que pertenece un determinado dato de entrada $\boldsymbol{x}_i$, y cómo se puede cuantificar el nivel de correspondencia de esta predicción con la clasificación real y_i mediante una función de pérdida.

El objetivo del entrenamiento es encontrar la configuración de los parámetros $\boldsymbol{W}$ capaz de clasificar correctamente todos (o casi todos) los datos de entrenamiento. Para ello, se recurrirá a un proceso de minimización de la función de pérdida, en el que se irán modificando los parámetros $\boldsymbol{W}$ de tal forma que según avance el entrenamiento, la pérdida disminuya.

La funciones de pérdida generalmente están definidas en espacios de muchas dimensiones, que coincidien en número con la cantidad de parámetros en nuestro modelo, por lo que es difícil visualizarlas espacialmente. Para unos datos de entrada concretos $\boldsymbol{x}_i$, si se realiza un corte en este espacio mediante un plano (bidimensional), se observaría que cada punto del plano, correspondiente a unos valores concretos de la matriz de pesos $\boldsymbol{W}$, presentará un valor de la función de pérdida.

Si se representa el valor de $\mathcal{L}$ para los puntos de este plano gráficamente, asignando una escala de colores a cada uno de los puntos según su valor, se obtendría un paisaje de la función de pérdida similar al representado en la figura 2.4.

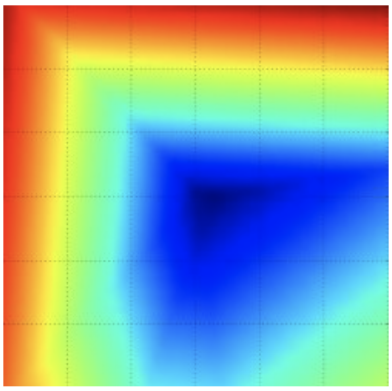

Figura 2.4: Ejemplo visual de un corte por un plano bidimensional en la función de pérdida. Se observa que van a existir zonas elevadas, de mucha pérdida (rojo), y otras más profundas, de poca pérdida (azul). El objetivo de la optimización será avanzar poco a poco hacia las zonas más profundas de la función de pérdida. Imagen tomada de (Karpathy, 2020).

Si se representase en tres dimensiones la figura 2.4, se obtendría una superficie convexa, siendo la zona más profunda la correspondiente a los colores en tonos azules, con poca pérdida. Si se partiera desde una zona elevada (en rojo) con mucha pérdida, la minimización consistiría en avanzar por la superficie siguiendo la pendiente hacia abajo, y de esta forma, alcanzar los valores mínimos de la función de pérdida. Extrapolando esta idea a espacios de mayor dimensión, si se parte de un punto cualquiera de la hipersuperficie de la función, la mejor forma de encontrar un nuevo punto con mejores parámetros, y por consiguiente, con pérdidas menores, es siguiendo la pendiente en sentido negativo. Para encontrar la dirección que presente la pendiente más pronunciada será necesario estudiar el *gradiente* de la función en el punto de partida.

Para una función escalar tal que $f : \mathbb{R}^D \to \mathbb{R}$, el gradiente consiste en un vector de dimensión D que determina la pendiente que presenta la superficie de la función en un determinado punto. Cada una de las componentes del vector gradiente se denomina *derivada parcial*.

$$\nabla f(\boldsymbol{z}) = (\frac{\partial f(\boldsymbol{z})}{\partial z_1}, \frac{\partial f(\boldsymbol{z})}{\partial z_2}, \ldots, \frac{\partial f(\boldsymbol{z})}{\partial z_D}) \tag{2.6}$$

La ecuación (2.6) muestra la forma que toma el gradiente de una función $\nabla f(\boldsymbol{z})$, evaluada en $\boldsymbol{z}$, donde $\frac{\partial f(\boldsymbol{z})}{\partial z_i}$ representa la derivada parcial de la función con respecto a la variable z_i.

Para el caso de una función de pérdida de grandes dimensiones $\mathcal{L}$, sabiendo que los datos de entrada $\boldsymbol{x}_i$ son dados y fijos, lo que interesa es conocer cómo varía $\mathcal{L}$ al modificar los parámetros $\boldsymbol{W}$. Esto implica que la derivada parcial que se necesita calcular es $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$, que estudiará la pendiente para cada uno de los pesos.

Hay varias formas de calcular el gradiente de una función en un punto, como pueden ser los métodos analíticos y los métodos numéricos. En los algoritmos clasificadores generalmente se emplea un método conocido como *propagación inversa*, que consiste en emplear la *regla de la cadena* analíticamente y calcularla de manera eficiente usando cálculos intermedios usados en la propagación directa (Karpathy, 2020).

Para una función de puntuación de asignación lineal $f(\boldsymbol{x}_i, \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}_i$ y una función de pérdida de entropía cruzada $\mathcal{L}_i$, el cálculo del gradiente $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$ por propagación inversa toma la forma

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} = \frac{\partial \mathcal{L}_i}{\partial f}\frac{\partial f}{\partial \boldsymbol{W}} + \frac{\partial}{\partial \boldsymbol{W}}(\lambda R(\boldsymbol{W})). \tag{2.7}$$

Una vez se conocen las derivadas parciales de las funciones, el cálculo de gradiente consiste simplemente en sencillas operaciones matriciales, como se mostrará en apartados posteriores.

Una vez se ha definida la forma en la que el algoritmo calcula los gradientes, es necesario definir la metodología que se va a seguir la minimización. El método de optimización por excelencia en los algoritmos clasificadores es el conocido como *descenso de gradiente*. Consiste en un proceso iterativo: inicialmente se calcula el gradiente en el punto actual; luego se avanza una determinada magnitud en esa dirección y en sentido negativo; posteriormente, se actualizan los valores de los parámetros para el nuevo punto; y finalmente, se calcula el gradiente en el nuevo punto, comenzando de nuevo el proceso. De esta forma se logra descender siempre en el sentido negativo del gradiente.

El descenso de gradiente en su versión más sencilla toma la forma:

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}} \tag{2.8}$$

Donde $\boldsymbol{W}_t$ es el valor actual de los parámetros; $\boldsymbol{W}_{t+1}$ es el valor actualizado; y η lo que se conoce como *magnitud del paso* o *tasa de aprendizaje*. El término $-\eta\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$ puede interpretarse como el incremento en los parámetros $\boldsymbol{W}$ al avanzar una magnitud η en la dirección del gradiente negativo.

Si se concibe la superficie de la función de pérdida como un mapa topográfico tridimensional (véase la figura 2.4), para una tasa de aprendizaje pequeña se avanzará con seguridad por la superficie hacia el gradiente negativo pero, al tomar pasos tan pequeños en cada iteración, es posible que el sistema se quede *atascado* en mínimos locales de la función, haciendo que el modelo no aprenda. Si en cambio la tasa de aprendizaje es demasiado grande, se avanzará a gran velocidad hacia el mínimo global, pudiendo suceder que, aunque se alcance un mínimo lo suficientemente bueno, al avanzar una magnitud elevada en el sentido del gradiente negativo *se pase de largo* por el mínimo, haciendo que el sistema no logre concluir el aprendizaje.

La tasa de aprendizaje es, por tanto, uno de los parámetros más delicados de los algoritmos clasificadores. Al depender la superficie de la función de variables multidimensionales, es difícil encontrar la magnitud del paso adecuada que permita al modelo aprender correctamente.

El aprendizaje se realiza de manera simultanea para muchos datos de entrenamiento. Cuando se trabaja con un conjunto demasiado grande, se requiere una elevada capacidad computacional para realizar la propagación inversa y calcular la actualización de los parámetros $\boldsymbol{W}$. Debido a esto, generalmente se trabaja con *lotes* o *mini-lotes* de datos de entrenamiento.

Esto funciona bien ya que el gradiente de la función de pérdida, al emplear un lote, es una buena aproximación del gradiente en el caso de emplear todos los datos de una sola vez. Cada lote se aprenderá durante un número finito de pasos t denominado *época*, quedando entonces el entrenamiento definido para un número de épocas, y no de pasos. Cuando el tamaño del lote es de únicamente un dato, el proceso se denomina *descenso de gradiente estocástico*.

La figura 2.5 muestra la dinámica del descenso de gradiente para los tres casos presentados: descenso de gradiente por lotes, descenso de gradiente por mini-lotes y descenso de gradiente estocástico.

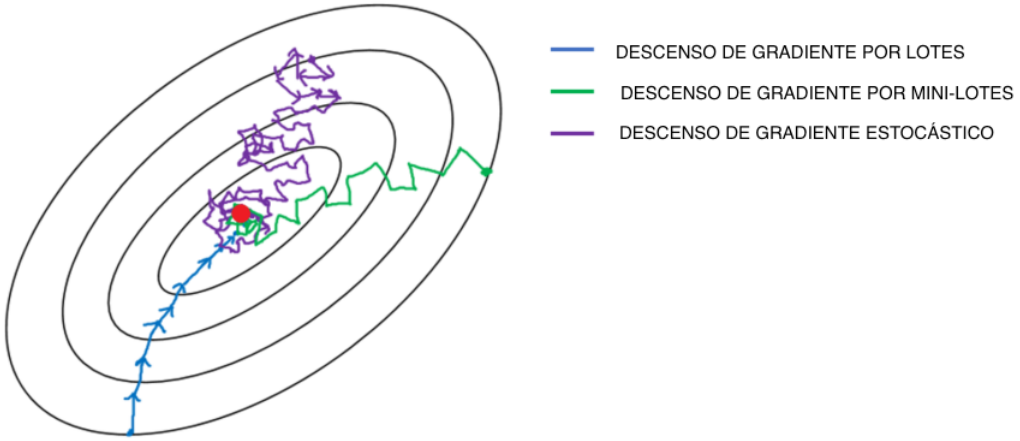

Figura 2.5: Ejemplo gráfico de cómo sería el descenso de gradiente por lotes, por mini-lotes y estocástico. Imagen tomada de (Dabbura, 2017).

2.1.4. Precisión

Mediante el descenso de gradiente se irán actualizando los valores de los parámetros $\boldsymbol{W}$. La idea es que el entrenamiento tome un número finito de pasos t en búsqueda de la menor pérdida posible, y que al finalizar, los parámetros obtenidos nos permitan clasificar otras imágenes similares a las empleadas durante el entrenamiento mediante $f(\boldsymbol{x}_i, \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}_i$.

Una forma de evaluar el rendimiento del entrenamiento es comparando, para cada paso t que se tome durante la optimización, el número de datos de entrada que se han clasificado correctamente. Si esto lo expresamos en porcentaje, se obtiene lo que se conoce como *precisión*.

La *precisión de entrenamiento* informa sobre cómo de bien está aprendiendo el modelo los datos de entrenamiento y la *precisión de evaluación* nos va a indicar cómo de bien clasifica el modelo unos datos con los que no había trabajado hasta entonces. A estos datos se les conoce como *datos de evaluación*.

La precisión de entrenamiento debe ser lo mayor posible al final del entrenamiento. Su evolución temporal da una idea de los mecanismos de aprendizaje y ayuda a la hora de realizar ajustes en el modelo. Para hacer modificaciones en el modelo, se suele emplear un nuevo conjunto de datos con los que el modelo tampoco ha trabajado hasta entonces, conocidos como *datos de validación* (Karpathy, 2020).

Se pueden modificar los hiperparámetros del modelo para alcanzar una elevada precisión de validación con los datos de validación, pero no se deben realizar ajustes para incrementar la precisión de evaluación.

Una elevada precisión de evaluación es la prueba definitiva que demuestra si un modelo es válido o no. Si se realizaran ajustes en función de la precisión de evaluación, se estaría tratando de ajustar la realidad al modelo y no al revés, por lo que el modelo no sería válido.

2.2. Redes Neuronales Artificiales

Tras conocer cómo aprenden y clasifican los clasificadores lineales, es importante también conocer sus limitaciones.

Dados unos datos de entrada $\boldsymbol{x}_i \in \mathbb{R}^D$, éstos estarán repartidos en un espacio de dimensión D. La función de puntuación de asignación lineal $f(\boldsymbol{x}_i, \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}_i$ los clasificará entonces según se sitúen con respecto a las regiones definidas por $f(\boldsymbol{x}_i, \boldsymbol{W}) = 0$.

El problema es que los límites de estas regiones son hipersuperficies planas. Puede existir una configuración de los datos en el espacio que sea demasiado compleja como para clasificarlos linealmente, por lo que sería necesario que los límites de las regiones correspondientes a cada clase fuesen superficies más complejas.

Las *redes neuronales artificiales* son capaces de definir regiones que se ajusten mejor a los datos de entrada. Para ello hacen uso de los propios clasificadores lineales, pero introducen en el modelo lo que se conoce como una *no-linealidad* (Karpathy, 2020).

En la imagen 2.6 se puede observar como en un espacio de dos dimensiones, para unos datos de entrada dispuestos en espiral agrupados en tres clases diferentes, un clasificador lineal (izquierda) no es capaz de aprenderlos correctamente. Al emplear una red neuronal (derecha), se observa como la clasificación es notablemente más precisa, por lo que se puede considerar un modelo válido.

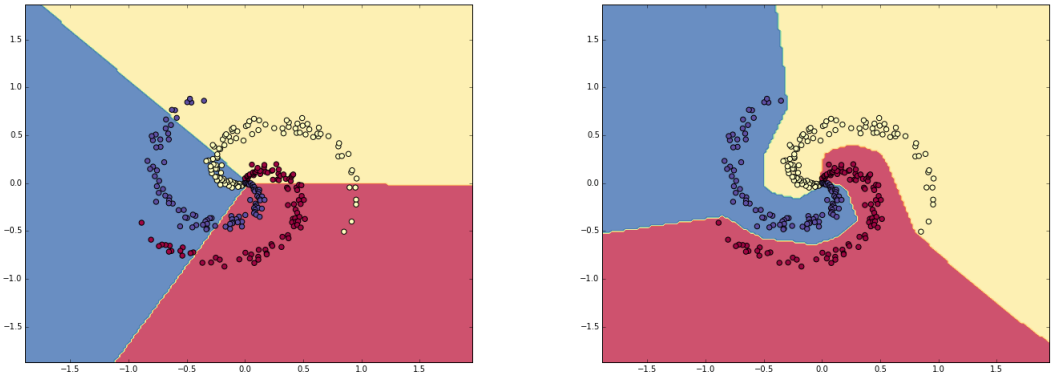

Figura 2.6: Regiones asignadas a cada clase en un problema de clasificación de los puntos de una espiral según su color. A la izquierda: un clasificador lineal no puede clasificar correctamente muchos de los puntos, debido a que presentan una distribución demasiado compleja. A la derecha: una red neuronal, en cambio, si es capaz de clasificarlos correctamente. Imagen tomada de (Karpathy, 2020).

2.2.1. Arquitectura

Una red neuronal artificial consiste en un conjunto de *unidades de procesamiento*, comunmente denotadas como *neuronas*, dispuestas en capas interconectadas de tal forma que la salida de unas neuronas sea la entrada de otras.

Dentro de cada unidad, se realiza una asignación mediante una función de puntuación lineal, como se ha visto hasta ahora. Por ejemplo, con la función $f(\boldsymbol{x}_i, \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}_i$. La diferencia está en que esta puntuación se evalúa, posteriormente, en una función no-lineal denominada *función de activación*, siendo el resultado obtenido la salida de la unidad.

La imagen 2.7 presenta de manera esquematizada la estructura de una red neuronal de tres capas, donde la capa correspondiente a los datos de entrada no se numera. Las capas que se encuentran entre la capa de entrada y la de salida se denominan *capas ocultas*. De manera general, es en estas capas donde las neuronas presentan la función de activación, ya que en la capa de salida lo que se quiere obtener es la puntuación real asignada a los datos de entrada.

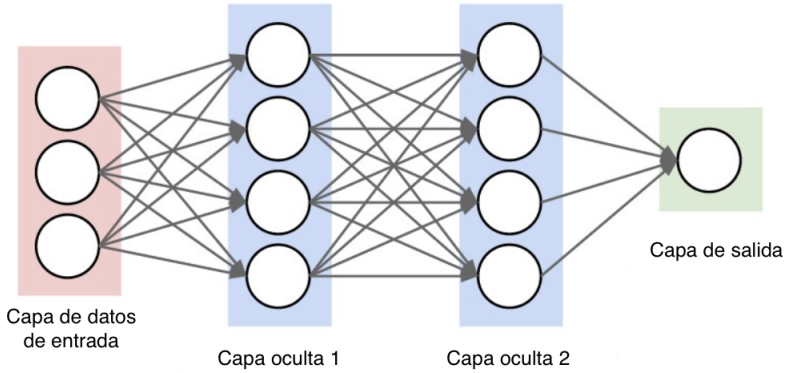

Figura 2.7: Esquema de una red neuronal de 3 capas, con dos capas ocultas completamente conectadas. Imagen tomada de (Karpathy, 2020).

Se denominará *profundidad* de la red al número de capas que presente, y *ancho* al número de neuronas que presente cada capa. La red neuronal de la imagen 2.7 presenta una configuración de capas *completamente conectadas*, en la que las neuronas de capas adyacentes están completamente conectadas dos a dos, sin embargo, dentro de cada capa las neuronas no se conectan entre sí.

La salida de una capa va a multiplicarse por una matriz de pesos $\boldsymbol{W}_i$ antes de pasar a la siguiente capa. De esta forma, la salida de la red neuronal para unos datos de entrada $\boldsymbol{x}$, con tres capas y una función de activación conocida como *ReLU*, que será detallada más adelante, tomará la forma $\boldsymbol{s} = \boldsymbol{W}_3 max(0, \boldsymbol{W}_2 max(0, \boldsymbol{W}_1 \boldsymbol{x}))$.

Al igual que la matriz de pesos $\boldsymbol{W}$ de los clasificadores lineales, las matrices $\boldsymbol{W}_1$, $\boldsymbol{W}_2$, $\boldsymbol{W}_3$ deberán optimizarse con descenso de gradiente para que la red neuronal aprenda los datos de entrenamiento.

2.2.2. Configuración de la red

Cuando una red aprende sin problemas los datos de entrenamiento, se habla de una red *sobreparametrizada*, y cuando no logra aprenderlos por no tener suficientes parámetros, se dice que está *subparametrizada*.

La arquitectura de una red neuronal va a determinar su desempeño frente a una determinada tarea. En este apartado se mencionarán las configuraciones de los hiperparámetros de la red más comunes.

2.2.2.1 Profundidad y ancho

A medida que se aumenta la profundidad y el ancho de una red, se está aumentando su capacidad. Esto se debe a que existe una mayor colaboración de las neuronas, por lo que pueden definir regiones en el espacio de mayor complejidad. A pesar de ello, elevar demasiado la capacidad puede tener consecuencias negativas para el modelo.

Generalmente, los conjuntos de datos de entrenamiento contienen casos aislados o ruido. Estos son, datos que se ubican en el espacio cerca de regiones que no corresponden a su clase, por lo que es más difícil clasificarlos correctamente.

Cuando una red presenta elevada capacidad, puede definir regiones en el espacio muy complejas que clasifiquen correctamente los datos, incluido el ruido. El problema de esto es que el ruido no representa las propiedades reales de los datos, por lo que al evaluar nuevos datos de entrada, las regiones definidas no darán buenos resultados de clasificación.

En esta situación se presentará una elevada precisión de entrenamiento pero una baja precisión de evaluación, diciéndose entonces que el modelo está *sobreajustado* (del inglés, *overfitted*). Es importante que los modelos presenten una *generalización* de los datos de entrenamiento, prestando menos atención al ruido y más a las grandes agrupaciones de datos a la hora de definir las regiones.

La aproximación clásica para corregir el sobreajuste a los datos de entrenamiento es reduciendo la capacidad de la red. Sin embargo, en la actualidad se ha comprobado que hay un límite por el cual al seguir aumentando el tamaño de la red mejora la generalización, contraintuitivatmente. Este fenómeno se conoce como *doble descenso* .

2.2.2.2 Función de activación

La función de activación es la clave del aprendizaje, ya que es la que va a introducir la no-linealidad propia de las redes neuronales. Algunas de ellas son:

- Función *Sigmoid*. Toma la forma $\sigma(x) = \frac{1}{1+e^{-x}}$. Reduce la entrada real a unos valores acotados en el intervalo [0,1], concretamente, los valores negativos grandes los convierte en 0 y los valores positivos grandes los transforma en 1.
- Función *Tanh*. Se define como $tanh(x) = 2\sigma(2x) - 1$. Reduce la entrada real a unos valores acotados en el intervalo [-1,1], por lo que quedan centrados con respecto al 0.
- Función *ReLU*. Se define como $f(x) = max(0, x)$, que simplemente convierte en cero los valores de entrada negativos.

La función ReLU es actualmente la función de activación por más popular y usada (Karpathy, 2020), ya que presenta una convergencia acelerada al emplear el descenso de gradiente. Durante el aprendizaje con esta función, se observa que algunas neuronas *mueren*, es decir, que llega un momento a partir del cual siempre dan como salida el valor 0. Este fenómeno depende principalmente de la tasa de aprendizaje y hace que la función ReLU sea notablemente delicada.

En la figura 6.2 se muestra la representación gráfica de las tres funciones de activación presentadas.

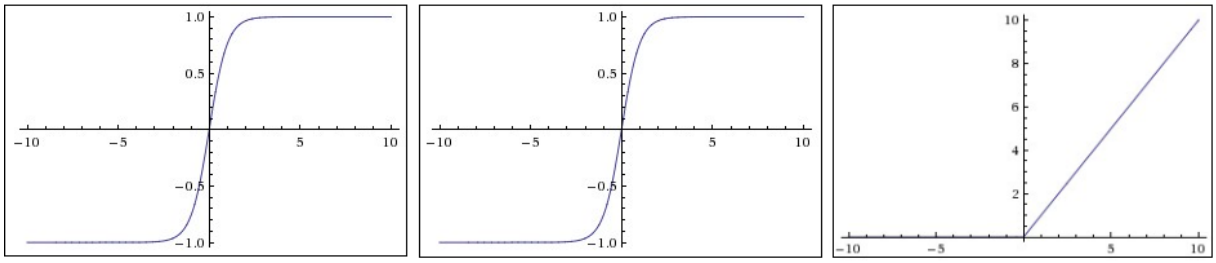

Figura 2.8: A la izquierda: la función de activación Sigmoid. En el centro: la función de activación tanh. A la derecha: la función de activación ReLU. Imagen tomada de (Karpathy, 2020).

2.2.2.3 Función de pérdida

En los problemas de clasificación se emplea, de manera general, la pérdida de entropía cruzada o la pérdida de bisagra, presentadas en apartados anteriores. También es común utilizar la penalización de regularización $L2$. Esto no se aplica únicamente a los clasificadores lineales, sino también a las redes neuronales.

A la hora de calcular la pérdida de regularización, es importante definir correctamente el parámetro de fuerza λ, ya que cuanto mayor sea este parámetro, mayor será la regularización. Una regularización elevada supone una mayor generalización del modelo. Por lo tanto, las redes suelen configurarse con elevada capacidad y, posteriormente, se define una fuerza de regularización suficiente para que contrarreste el sobreajuste.

2.2.2.4 Método de optimización

El descenso de gradiente se aplica generalmente en su versión más sencilla, presentada en la ecuación (2.8). En redes neuronales es común emplear también una actualización de parámetros conocida como *actualización de momento*.

Este tipo de actualización presenta una inercia que recuerda pasos anteriores al descender el sistema por la superficie de la función de pérdida. Esto permite *frenar* el sistema según se acerca a un mínimo y que no se vea tan afectado por cambios pequeños en la superficie. Un esquema popular en redes neuronales profundas es la actualización mediante *momento Nesterov*, que combina el gradiente en el punto actual con pasos anteriores.

2.2.2.5 Tasa de aprendizaje

De manera general, la tasa de aprendizaje se define como constante durante todo el entrenamiento. Al avanzar una magnitud constante en cada paso del aprendizaje, se corre el riesgo de no poder alcanzar un mínimo lo suficientemente profundo. Existen otros métodos que reducen la magnitud del paso según se avanza en el entrenamiento:

- Caída exponencial. Se expresa matemáticamente como $\eta = \eta_0 e^{-kt}$.
- Caída 1/t. Se define como $\eta = \eta_0/(a + kt)$.

donde η_0 es la tasa actual; k y a son parámetros a definir; y t es el numero de paso (o de época).

2.2.3. Redes neuronales convolucionales

Las redes neuronales artificiales que se han visto hasta ahora trabajan muy bien con conjuntos de datos sencillos y generalmente se emplean a nivel académico. Las *redes neuronales convolucionales*, en cambio, son una versión más compleja y especializada de éstas: son las que se emplean a nivel profesional en tareas de clasificación de imágenes, como puede ser el reconocimiento facial o en la detección de objetos en la conducción autónoma.

Las redes convolucionales son muy similares a las redes neuronales convencionales: tienen como objetivo clasificar un conjunto de datos determinando el valor adecuado de los parámetros $\boldsymbol{W}$. También están constituidas a partir de capas de neuronas capaces de introducir una no-linealidad a la clasificación. La diferencia reside en la forma en la que se disponen estas neuronas (**karpathy**).

Las redes convolucionales están especializadas en la clasificación de imágenes. Es por ello que mantienen la imagen como una matriz tridimensional de píxeles, en vez de transformarla en un vector. Esto hace que tanto sus capas ocultas como la de salida presenten una distribución de neuronas en tres dimensiones: *ancho*, *altura* y *profundidad*. La imagen 2.9 muestra un esquema muy simplificado de cómo sería una red convolucional.

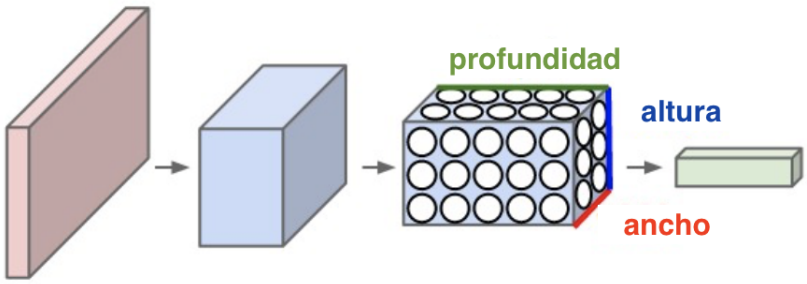

Figura 2.9: Esquema simplificado de una red neuronal convolucional de 3 capas, con dos capas ocultas. Imagen tomada de (Karpathy, 2020).

Este tipo de redes suelen estar formadas por un gran número de capas interconectadas. Presentan distintos tipos de capa que cumplen una determinada función en el proceso de aprendizaje:

- Capa de datos de entrada. Es la capa que contiene los píxeles sin transformar de la imagen. Por ejemplo, tendrá una dimensiones de [32x32x3]: 32 píxeles de alto, 32 de ancho y los 3 canales de color RGB .
- Capa convolucional . En esta capa se encuentra un conjunto de neuronas, cada una de las cuales tiene asignada una región de la imagen, que calculará una puntuación para su región. Por ejemplo, la imagen puede dividirse en 12 regiones, de forma que esta capa tiene una dimensión [32x32x12].
- Capa ReLU. Esta capa tiene como salida el resultado de aplicar la función de activación ReLU, por lo que la dimensión sigue siendo [32x32x12].
- Capa de agrupación. Esta capa tiene como objetivo reducir la resolución de los datos, por ejemplo, pasando de [32x32x12] a [16x16x12].

- Capa de salida. Generalmente se trata de una capa completamente conectada, que tendrá como salida la puntuación final asociada a cada clase para la imagen de entrada, y tendrá una dimensión [1x1xk], siendo k el número de categorías.

Estas capas pueden distribuirse de muchas maneras según el problema de clasificación propuesto. Un ejemplo de configuración de las capas puede ser [ENT-CONV-RELU-CONV-RELU-AGR-CONV-RELU-CONV-RELU-AGR-CONV-RELU-CONV-RELU-AGR-SAL], siendo la correspondencia de acrónimos: ENT para la capa de entrada, CONV para la capa convolucional, RELU para la capa ReLU, AGR para la capa de agrupación y SAL para la capa de salida.

3. La función de pérdida dinámica

3.1. Motivación

Uno de los grandes retos del aprendizaje automático en los últimos años ha sido mejorar el rendimiento de las redes neuronales.

Por ejemplo, se ha demostrado como una correcta inicialización de los parámetros logra resultados de aprendizaje más eficientes. En esta línea de investigación destaca el trabajo de Glorot y Bengio, quienes presentaron una inicialización normalizada de los parámetros $\boldsymbol{W}$ y se estudió su influencia al emplear distintas funciones de activación, demostrándose que lograba una convergencia más rápida (Glorot y Bengio, 2010). También se han estudiado esquemas de inicialización de parámetros en redes convolucionales profundas, del orden de miles de capas, que lograban reducir considerablemente el número de capas necesarias para el aprendizaje (Xiao et al., 2018).

Otras líneas de investigación se han centrado en la arquitectura de la red. Las denominadas *redes neuronales residuales*, una variante de las redes convolucionales, establecen conexiones entre capas de la red que no son adyacentes, logrando así mejores resultados empleando un menor número de capas (He et al., 2016). Posteriormente, en otros trabajos (Zoph y Le, 2017), se empleó este tipo de redes para generar arquitecturas de redes neuronales eficientes.

Estudios recientes se han centrado en la optimización de la función de pérdida (Choromanska et al., 2015; Arous et al., 2019; Soudry y Carmon, 2016). Generalmente, las funciones de pérdida son funciones no-convexas: presentan una superficie con una gran cantidad de mínimos locales y puntos de silla, además de grandes variaciones en la curvatura. Esto hace que alcanzar el mínimo global de la función sea complicado en ciertos casos.

Existen mínimos locales lo suficientemente profundos como para que la red neuronal sea capaz de aprender. Aun así, sigue siendo un desafío alcanzarlos: la gran cantidad de inicializaciones, métodos de optimización y ajustes de hiperparámetros hace que sea difícil encontrar la combinación adecuada de los mismos que permita descender adecuadamente por la superficie de la función. Esto sugiere un nuevo planteamiento:

¿Es posible definir una función de pérdida que cambie durante su optimización para facilitar el aprendizaje?

El *aprendizaje curricular*, una estrategia de entrenamiento para redes neuronales introducida por Bengio *et al.*, consiste en presentar los datos de entrenamiento ordenados de tal forma que se muestre gradualmente cada vez más datos, y cada vez más complejos (Bengio et al., 2009). De una manera abstracta, el aprendizaje curricular puede entenderse como una secuencia de criterios de aprendizaje, en la que cada criterio presenta un conjunto de pesos asociado a los datos de entrenamiento. Inicialmente, los pesos van a favorecer el aprendizaje de los datos definidos como sencillos; al pasar al siguiente criterio, se va a cambiar ligeramente la distribución de pesos para aumentar la probabilidad de muestrear datos más complejos; al final de la secuencia, el conjunto de pesos tendrá una distribución uniforme que permita trabajar con la totalidad del conjunto de datos. De esta forma la función de coste va cambiando progresivamente durante el entrenamiento, facilitando el apredizaje.

La implementación del aprendizaje curricular permite lograr una convergencia más rápida a mejores soluciones y pudiéndose obtener modelos de mayor generalización. A pesar de ello, uno de los principales retos de esta metodología es definir el nivel de dificultad de los datos de entrenamiento, imposible en algunos casos.

Diversos problemas físicos se reducen a la optimización de una superficie de potencial (Ruiz-Garcia et al., 2017; Ruiz-Garcıa et al., 2016; Cea et al., 2019; Morshedifard et al., 2021). En *Tuning and jamming reduced to their minima* (Ruiz-García et al., 2019), se estudió la optimización de una red de flujo inspirada por la estructura de vasos sanguíneos en el cerebro. El objetivo de la investigación era tratar de simplificar el problema de optimización modificando la función a optimizar. Este trabajo inspiró la investigación que presentamos en este Trabajo Fin de Grado. El sistema de la red de flujo estaba constituido por un conjunto de nodos, con una presión escalar asociada a cada uno, unidos mediante conductos a través de los cuales pasa una corriente. Dada una red inicial y un conducto *fuente*, el problema consistió en optimizar la red añadiendo o eliminando enlaces de tal forma que se lograse alcanzar una determinada diferencia de presión en unos conductos *objetivo*. Se observó que ajustar la caída de presión de manera ordenada, según la distancia entre el conducto fuente y los objetivo, ayudaba a la optimización. Definir la caída de presión de un conducto distante afectaba de manera significativa al resto de la red, haciendo más difícil la calibración posterior de otros conductos. En cambio, definir la caída de presión en un conducto próximo al fuente permitía ajustar con mayor facilidad la presión en otros conductos más distantes.

Se introdujo entonces un factor en la optimización, función de esta distancia, que favorecía el ajuste de la presión de manera ordenada. Se llegó a la conclusión de que esta pequeña modificación lograba reducir la dificultad del problema, cambiando la topografía de los mínimos locales de la función sin afectar al mínimo global, y que podría aplicarse de manera similar en algoritmos de redes neuronales.

Combinando el aprendizaje curricular y el empleo de factores selectivos en la función de pérdida, surgió lo que se conoce como la *función de pérdida dinámica*. A lo largo de este apartado se realizará un estudio detallado de esta función, mostrando las ventajas que proporciona su implementación y estudiando sus mecanismos de aprendizaje.

3.2. Introducción

En *Tilting the playing field: Dynamical loss functions for machine learning* (Ruiz-García et al., 2021) se presentó por primera vez el concepto de función de pérdida dinámica. Ésta función consiste en transformar una función de pérdida convencional para que modifique su topografía durante el entrenamiento, permitiendo que se exploren nuevas regiones del espacio de parámetros durante la optimización. De esta forma, es posible alcanzar mínimos más profundos que logren mejores resultados de aprendizaje.

De manera similar a la metodología del *aprendizaje curricular* (Bengio et al., 2009), Ruiz *et al.* desarrollaron un algoritmo capaz de aprender de manera gradual y siguiendo un determinado orden. En este sentido, en vez de definir una nueva categoría para clasificar los datos según el nivel de dificultad, se tomó ventaja de la división por clases preexistente.

Para un conjunto de datos previamente dividido en clases, la función de pérdida dinámica consiste en introducir en la función de pérdida unos factores dependientes del tiempo asociados a cada clase. Al entrenar la red, estos factores fomentan el aprendizaje de una de las clases sobre el resto durante un determinado periodo de tiempo y van oscilando de tal manera que se enfatiza una clase tras otra cíclicamente.

La figura 3.1 muestra de manera gráfica valor de los factores $\Gamma_i(t)$ en función del número de pasos tomados en el entrenamiento, para unos datos de entrenamiento divididos en tres clases, representando cada color el factor asociado a cada clase. Se puede observar que el tiempo total de simulación, de 20000 pasos, se divide a su vez en periodos de $T = 2500$ pasos. En cada periodo, el factor de una de las clases es notablemente superior a los demás. Al pasar al siguiente periodo, el valor de este factor decae, siendo ahora el factor asociado a la siguiente clase el que toma relevancia. Se observa que este proceso se repite de manera cíclica y siguiendo un orden preestablecido.

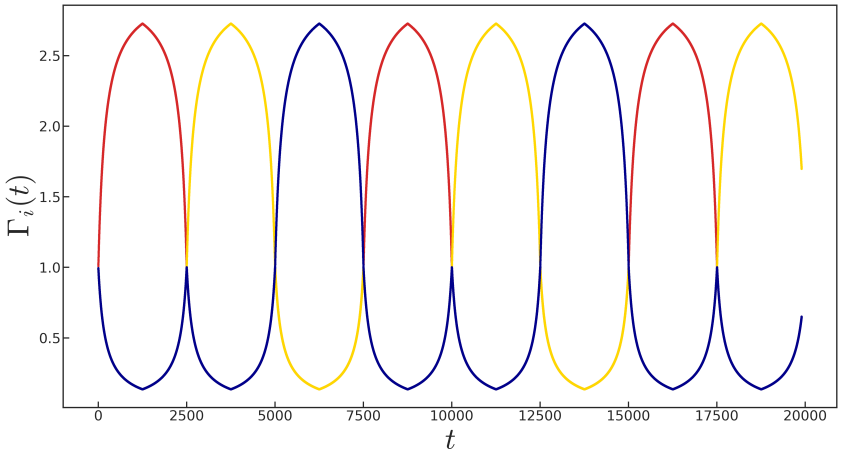

Figura 3.1: Ejemplo de la evolución temporal del valor de los factores asociados a cada una de las tres clases [*rojo*, *amarillo* y *azul*].

Al implementar esta transformación en redes subparametrizadas, que no logran alcanzar un mínimo lo suficientemente profundo en la *versión estática* de la función de pérdida, se logró aumentar tanto la precisión de entrenamiento como la de evaluación. Incluso en redes sobreparametrizadas, que ya presentaban la máxima precisión de entrenamiento, se logró aumentar la precisión de evaluación, logrando una mejor generalización del modelo.

Conceptualmente, se puede entender que la función de pérdida dinámica va cambiando la curvatura de su superficie durante el entrenamiento, de forma que, aunque se esté siempre descendiendo por la superficie instantánea de la función, el sistema va a cruzar determinadas barreras que limitaban su progreso al emplearse la función de pérdida convencional, en la que los pesos son uniformes para todas las clases.

El proceso es similar a un movimiento peristáltico, en el que de manera cíclica, los mínimos locales - o como se denotará a partir de ahora, los *valles de la superficie* - de la función se elevan y estrechan, para luego descender y ensancharse. Esto permite que el sistema alcance mínimos más profundos, además de más anchos, hecho que se ha demostrado que presenta una correlación con la generalización del modelo (Zhang et al., 2017).

Cuando se encuentra en regiones demasiado estrechas para la magnitud del paso establecido, el sistema es empujado a otras zonas de la superficie de la función. Un fenómeno característico de este proceso es la aparición de cascadas de bifurcaciones en la dinámica de optimización, las cuales facilitan el aprendizaje. En los apartados siguientes se explicará su presencia en relación con los autovalores de la matriz hessiana de la función y con los autovalores del Núcleo de la Tangente Neuronal (en inglés, *Neural Tangent Kernel*), una función que permite describir la evolución de la red neuronal durante el entrenamiento (Jacot et al., 2020).

3.3. Formulación

La función de pérdida dinámica consiste en introducir una función, dependiente del tiempo t en el que se encuentre el sistema durante la optimización, en la función de pérdida convencional. Esta transformación fue aplicada por primera vez en la función de pérdida de entropía cruzada (Ruiz-García et al., 2021).

$$\mathcal{F} = \sum_{j \leq P} \Gamma_{y_j}(t) \left(-\log \left(\frac{e^{f_{y_j}(\boldsymbol{x}_j, \boldsymbol{W})}}{\sum_i e^{f_i(\boldsymbol{x}_j, \boldsymbol{W})}} \right) \right) \tag{3.1}$$

La ecuación (3.1) muestra la implementación de la función de pérdida dinámica de entropía cruzada. Se asume un conjunto de datos P, en el que cada dato $\boldsymbol{x}_j$ pertenece a una clase y_j. La salida de la red neuronal dado un determinado dato de entrada $\boldsymbol{x}_j$ y una matriz de pesos $\boldsymbol{W}$ es $f(\boldsymbol{x}_j, \boldsymbol{W}) \in \mathbb{R}^C$, siendo C el conjunto de clases.

La superficie de la función de pérdida variará en función del valor de Γ_i, que se emplea para enfatizar el aprendizaje de una determinada clase sobre el resto durante un periodo T. Por lo tanto, se puede deducir que el tiempo que tarda el sistema en rotar de manera ordenada por todas las clases es CT. Para simplificar las ecuaciones, se define el tiempo dentro de cada periodo T como

$$t_T = t \text{ mód } T. \tag{3.2}$$

Se define una función que crece y luego decrece linealmente en función de t_T, con una amplitud A, de la forma

$$g(t_T) = \begin{cases} 1 + mt_T & \text{para} \quad 0 < t_T \leq T/2 \\ 2A - mt_T - 1 & \text{para} \quad T/2 < t_T \leq T, \end{cases} \tag{3.3}$$

siendo $A \geq 1$ la amplitud y $m = 2(A-1)/T$. Se definen a partir de *g(t)* unos factores asociados a cada clase

$$\hat{\Gamma}_i = \begin{cases} g(t_T) & \text{para} \quad t/T \text{ mód } C = i \\ 1 & \text{para} \quad t/T \text{ mód } C \neq i, \end{cases} \tag{3.4}$$

donde j es la clase en la que se enfatiza el aprendizaje. Si además normalizamos estos factores, se obtienen los factores Γ_i que se emplean en la ecuación (3.1):

$$\Gamma_i = C\frac{\hat{\Gamma}_i}{\sum_{j=1}^{C}\hat{\Gamma}_j}. \tag{3.5}$$

Para $A = 1$, el algoritmo aprende todas las clases por igual y, por lo tanto, la función no presenta oscilaciones; para $A \to \infty$, el algoritmo aprende únicamente una clase por periodo. Nótese que debido a la normalización, cuando Γ_i crece, $\Gamma_{j\neq i}$ decrece. Además, en cada periodo $\Gamma_i > 1$ para la clase i y $\Gamma_{j\neq i} < 1$ para el resto de las clases.

Es importante destacar que, a pesar de que se modifique la topografía de la función de pérdida, los mínimos globales a los que tiende la función durante la optimización siguen siendo los mismos. Este tipo de transformación está fundamentada en trabajos previos en los que se estudió el proceso de optimización de redes de flujo (Ruiz-García et al., 2019), que se presentaron en apartados anteriores.

3.4. Aplicaciones

3.4.1. *CIFAR-10* y *Myrtle5*

La función de pérdida dinámica de entropía cruzada fue inicialmente probada en una red neuronal convolucional, *Myrtle5*, empleando la base de datos de entrenamiento *CIFAR-10* (Ruiz-García et al., 2021).

La redes *Myrtle* fueron presentadas por Shankar *et al.* como una versión simplificada de una arquitectura para redes neuronales convolucionales desarrollada por la empresa *Myrtle.ai* (Myrtle.ai, 2018). Las distintas versiones de la arquitectura original fueron denotadas como *Myrtle5*, *Myrtle7* y *Myrtle10* según el número de capas de la red. Se trata de una familia de redes neuronales de arquitectura sencilla que logran un buen desempeño para la base de datos *CIFAR-10* (Shankar et al., 2020).

La figura 3.2 muestra los resultados obtenidos empleando una red *Myrtle5* con *CIFAR-10*. Para construir las gráficas, en cada punto se tomó el valor medio de la precisión de 30 simulaciones con los mismos hiperparámetros y con inicializaciones aleatorias de la matriz de pesos $\boldsymbol{W}$. Se empleó un descenso de gradiente estocástico como método de optimización, usándose 64 canales con un algoritmo de optimización Nesterov de momento $= 0,9$ y tomando las imágenes en lotes de 512. La simulación completa consta de 700 épocas, parándose las oscilaciones ($A = 1$) en la época 600. Se empleó una tasa de aprendizaje variable linealmente en el tiempo: empezando en 0, alcanza un máximo igual a $0,02$ en la época 300, para finalmente descender a 0,002 en la época final.

Para el caso $A = 1$, en el que se trabaja con la función de pérdida convencional, se observa en las gráficas que la red es capaz de aprender de manera perfecta los datos de entrenamiento (precisión de entrenamiento ~ 1). Esta red está sobreparametrizada, por lo que sin las oscilaciones, correspondiente a la función de pérdida estacionaria, ya es capaz de aprender todos los datos de entrenamiento. A pesar de ello, la precisión de validación toma valores más modestos ($\sim$ 0,73).

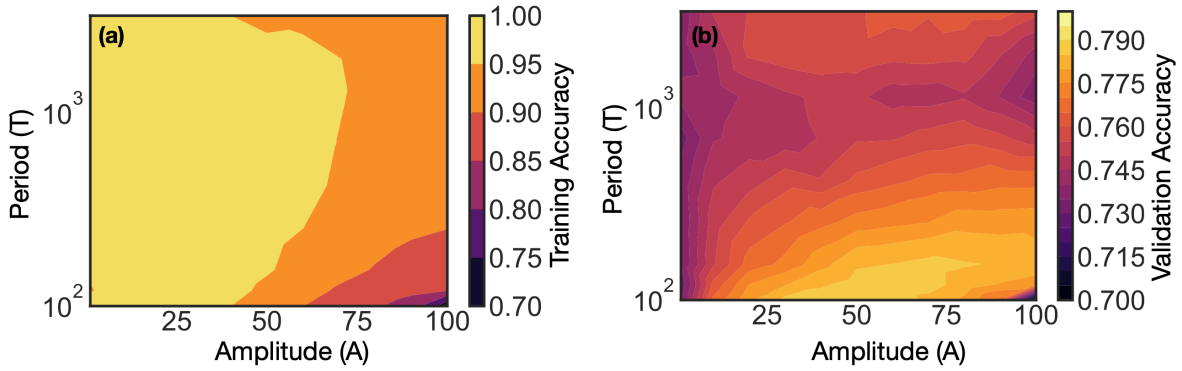

Figura 3.2: Gráficas de contorno de la función de pérdida dinámica de entropía cruzada, implementada en una red neuronal *Myrtle5* y con *CIFAR-10* como base de datos. La figura contiene dos paneles: (a) muestra la precisión de entrenamiento y (b) la precisión de evaluación, ambas en función de la amplitud (A) y el tamaño de periodo (T). Se empleó un descenso de gradiente estocástico como método de optimización,con un algoritmo de optimización Nesterov de momento $= 0,9$ y tomando las imágenes en lotes de 512. La simulación completa consta de 700 épocas, parándose las oscilaciones ($A = 1$) en la época 600. Se empleó una tasa de aprendizaje variable linealmente en el tiempo: empezando en 0, alcanza un máximo igual a $0,02$ en la época 300, para finalmente descender a 0,002 en la época final. Gráficas tomadas de (Ruiz-García et al., 2021).

En las gráficas se pueden encontrar regiones de los parámetros A y T en los que la función de pérdida dinámica alcanza mejores resultados que la convencional. En la región ($25 \lesssim A \lesssim 70$, $T \lesssim 250$) encontramos un incremento del $\sim 6\,\%$ en la precisión de validación que alcanza valores de $\sim 0,79$. Estos resultados mostraron que la función de pérdida dinámica incrementa la generalización de manera significativa (Zhang et al., 2017).

3.4.2. El caso de la espiral

Para entender mejor cómo la función de pérdida dinámica logra mejorar los resultados, Ruiz *et al.* emplearon como conjunto de datos de entrenamiento una distribución de puntos en espiral (Ruiz-García et al., 2021).

La figura 3.3 muestra una recreación del conjunto de datos, propuesto originalmente en (Karpathy, 2020). Cada uno de los tres brazos de la espiral constituye una clase de 100 puntos, siendo representada cada clase por un color. La trayectoria de los puntos sigue una distribución normal con una desviación típica de $\sigma = 0,2$.

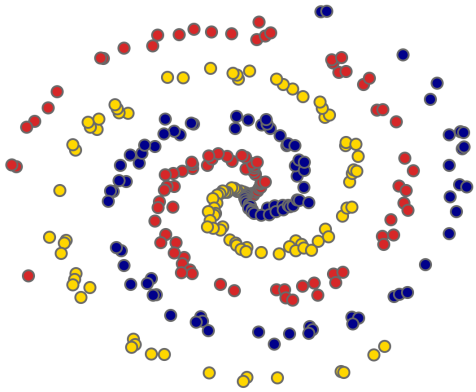

Figura 3.3: Recreación del conjunto de datos de entrenamiento en espiral empleado en (Ruiz-García et al., 2021).

Para estos datos de entrenamiento, se empleó una red neuronal de una única capa oculta y se ha optimizado mediante un descenso de gradiente de lote completo.[1] Se estudiaron los resultados para dos variantes de la misma red: una con una capa oculta de ancho igual a 100 neuronas y otra con una capa oculta de 1000 neuronas (Ruiz-García et al., 2021).

La figura 3.4 muestra unas gráficas de contorno similares a las presentadas en la figura 3.2. Los paneles (a) y (b) muestran, respectivamente, la precisión de entrenamiento y de evaluación en función de la amplitud (A) y el periodo (T) para la red de ancho igual a 100. Los paneles (c) y (d) muestran lo mismo para el caso de la red de ancho igual a 1000. Para cada punto de cada gráfico se ha tomado el valor medio de 50 simulaciones, en las que se ha optimizado con descenso de gradiente de lote completo con 35.000 pasos y con una tasa de aprendizaje constante e igual a 1. En todas las simulaciones se han parado las oscilaciones ($A = 1$) en el último periodo.

En el caso de la red de 100 neuronas (subparametrizada), con la función de pérdida convencional, correspondiente al caso en el que $A = 1$, la red no es capaz de aprender los datos de entrenamiento correctamente, alcanzando precisiones de entrenamiento y validación de $\sim 0{,}65$. Esto supone que la función de entropía cruzada presenta una topografía demasiado compleja para los hiperparámetros establecidos y, por lo tanto, el sistema no es capaz de encontrar un mínimo profundo.

En el caso en el que se están aplicando las oscilaciones ($A > 1$), existe una región en la gráfica de contorno ($5 \lesssim A \lesssim 20$, $T \lesssim 300$) donde la precisión de entrenamiento es casi perfecta y la precisión de evaluación alcanza valores de $\sim 0{,}9$.

Al igual que en el caso con *Myrtle5* y *CIFAR-10*, cuando se trabaja con una red lo suficientemente ancha, la precisión de entrenamiento alcanza valores de ~ 1 para el caso de la función de pérdida estática ($A = 1$), como es el caso de la red de ancho igual a 1000 (sobreparametrizada). De nuevo se puede observar que en una determinada región ($5 \lesssim A \lesssim 25$, $T \lesssim 700$) la precisión de evaluación es superior en la función de pérdida dinámica, hecho que implica una mayor generalización de los resultados.

[1] Cuando el conjunto de datos de entrenamiento es lo suficientemente pequeño, no es necesario planificar su aprendizaje por lotes; en cambio, se trabaja con *lote completo*, empleando todo el conjunto de datos.

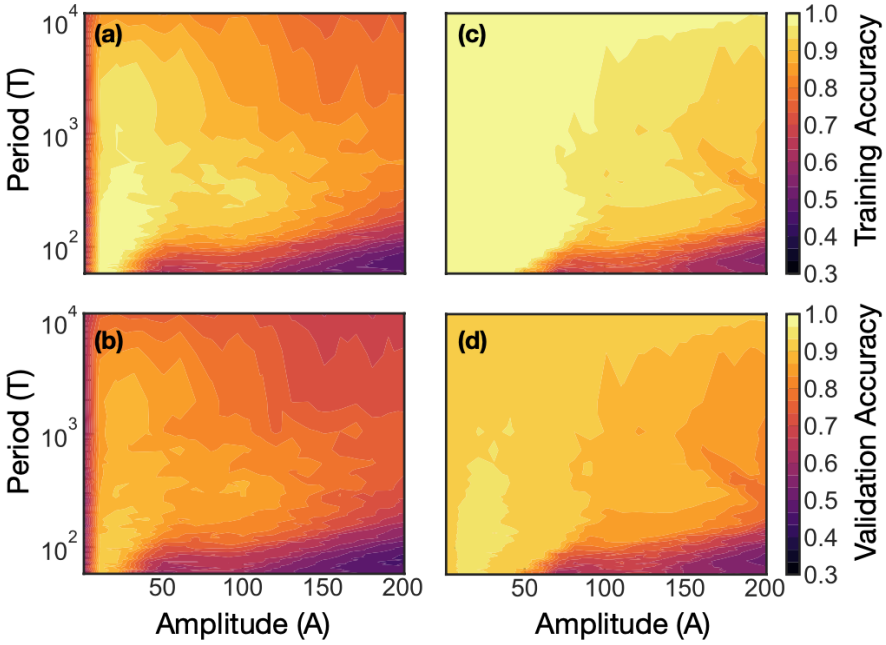

Figura 3.4: Gráficas de contorno de la función de pérdida dinámica de entropía cruzada, implementada en una red neuronal de una capa oculta y tomando una espiral tricolor análoga a la figura 3.3 como conjunto de datos. Se muestran los resultados de aprendizaje en una red de ancho 100 (paneles a y b) y otra de 1000 (paneles c y d), respectivamente. Para crear la gráfica se tomó el valor medio de 50 simulaciones en cada punto de rejilla en el plano (T, A). En cada simulación se empleó un descenso de gradiente de lote completo de 35000 pasos, una tasa de aprendizaje constate de valor 1 y se detuvieron las oscilaciones ($\Gamma_i = 1$) en el último periodo. Gráficas tomadas de (Ruiz-García et al., 2021).

3.5. Dinámica del aprendizaje

La optimización de la función de pérdida dinámica supone la coexistencia de dos fenómenos diferentes: la superficie de la función cambia según avanza el entrenamiento, al mismo tiempo que el sistema desciende en en cada instante tratando de alcanzar el mínimo de la función.

La dinámica del sistema durante el aprendizaje queda plasmada en las gráficas de la figura 3.5, simulaciones que se han llevado a cabo dentro de este trabajo de investigación y que son diferentes de las presentadas en los trabajos de Ruiz *et al.* (Ruiz-García et al., 2021). Se ha entrenado la red neuronal de una capa oculta tomando la espiral como conjunto de datos de entrenamiento, con $T = 5000$ y $A = 70$. Debido a que se trata de una simulación larga de 70000 pasos, se pueden tomar estos valores de periodo y amplitud *elevados*, ya que la red logra aprender correctamente los datos de entrenamiento (precisión ~ 1) y facilitan la visualización. Para simulaciones más cortas, se suelen tomar valores más pequeños de A y T, ya que arrojan mejores resultados (véase la figura 3.4).

En el panel (b) puede observarse la evolución temporal de la función de pérdida dinámica $\mathcal{F}(t)$. En cada periodo T la función toma una forma convexa, mostrando que inicialmente el sistema desciende por una región cada vez más favorable para la clase i que se está enfatizando, pero a medida que Γ_i disminuye, las otras clases toman relevancia y la región vuelve progresivamente a su topografía original.

La función de pérdida de entropía cruzada convencional, denotada como $\mathcal{L}(t)$, está representada en el panel (c). En este caso, en cada periodo la función toma una forma cóncava, debido a que en esta función todas las clases computan de manera equitativa a la pérdida. Al descender el sistema por una región que favorece a una clase frente a las otras, desde el punto de vista de la función convencional, el sistema esta en una región que no clasifica correctamente dos de las tres clases de datos y por lo tanto la pérdida irá incrementando según se descienda en ese sentido.

A pesar de ello, se observa que el sistema logra en cada periodo alcanzar valores cada vez más pequeños de la función de pérdida, hecho que se traduce en la precisión de entrenamiento (panel e). En el último periodo, en el que se han eliminado las oscilaciones, ambas funciones desembocan en el mismo valor.

En los paneles (b-f) se puede comprobar que a lo largo del proceso de aprendizaje el sistema se encuentra con situaciones de inestabilidad, que se manifiestan en forma de cascadas de bifurcaciones. Sorprendentemente, estas inestabilidades no solo no suponen una amenaza para el aprendizaje, sino que, incluso, se considera que favorecen el éxito del entrenamiento (Ruiz-García et al., 2021).

La matriz hessiana, al describir la curvatura de la superficie de la función en cada instante, puede ayudar a entender el proceso de aprendizaje y el origen de las inestabilidades. En el panel (f) se encuentran los tres mayores autovalores de la Hessiana, así como el umbral del mayor de los autovalores, a partir del cual el sistema se desestabiliza. Lo que ocurre es que al llegar el sistema a una región de la superfice que es demasiado estrecha para la tasa de aprendizaje, la dinámica de optimización se desestabiliza y el sistema *rebota* entre las dos paredes del valle que está descendiendo.

Por último se representa en el panel (g) el mayor autovalor del Núcleo de la Tangente Neuronal, una función que permite entender la evolución del sistema durante el entrenamiento. Se puede observar como el valor de λ_{max} decrece drásticamente durante las inestabilidades, tratando de *contrarrestarlas*.

Estos dos últimos paneles (f y g) serán claves a la hora de explicar el proceso de aprendizaje de la función de pérdida dinámica y los explicamos en más detalle en los siguientes apartados.

3.5.1. La matriz hessiana

La matriz hessiana muestra de manera ordenada todas las derivadas parciales de segundo orden de una función, las cuales indican cómo varían las derivadas primeras con respecto a las variables de entrada (Goodfellow et al., 2016).

Para una función escalar, tal que

$$\begin{aligned} f\colon \mathbb{R}^n &\longrightarrow \mathbb{R} \\ \boldsymbol{x} &\longmapsto f(\boldsymbol{x}), \end{aligned}$$

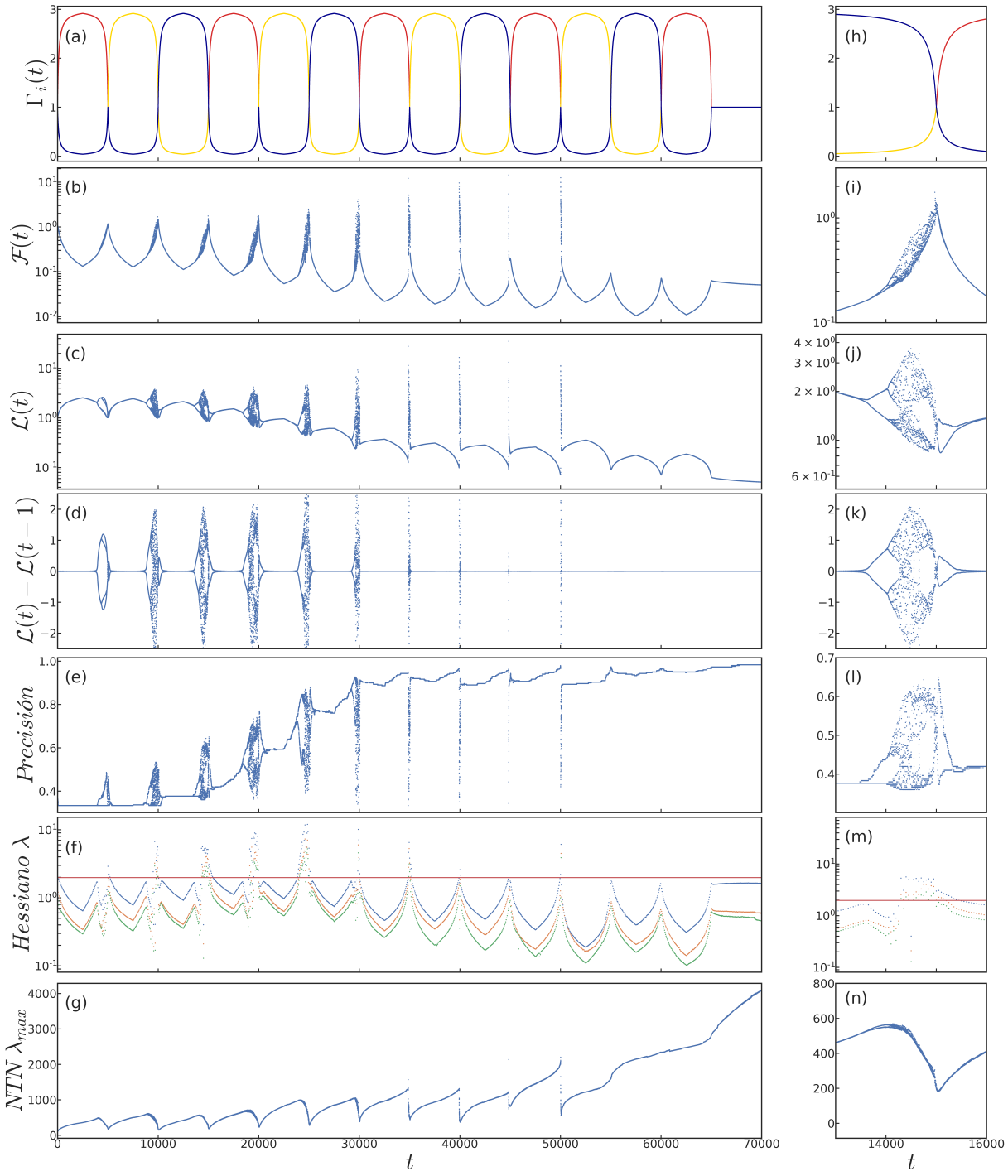

Figura 3.5: Dinámica del aprendizaje al emplear la función de pérdida dinámica (3.1). En este caso se empleó una red neuronal de una capa oculta de ancho 100, tomando una amplitud de $A = 70$ y un periodo de $T = 5000$, empleando un conjunto de datos en espiral, similar al presentado en la figura 3.3. El panel (a) muestra Γ_i según avanza t, estando cada una de las tres clases representada por un color, al igual que en 3.1. En el panel (b) se representa el valor de la función de pérdida dinámica $\mathcal{F}(t)$. El panel (c) muestra el valor de la función de pérdida convencional $\mathcal{L}(t)$, en la que para todas las clases $\Gamma_i = 1$. En el panel (d) se representa $\mathcal{L}(t) - \mathcal{L}(t-1)$ para exponer de manera clara las inestabilidades. El panel (e) muestra el valor de la precisión durante el entrenamiento. En el panel (f) se representa el valor de los tres mayores autovalores de la Hessiana de la función de pérdida dinámica. En este panel también se ha pintado en rojo el umbral de λ_{max} a partir del cual se desestabiliza el sistema. El panel (g) representa la evolución temporal del autovalor máximo del Núcleo de la Tangente Neuronal (Jacot et al., 2020). Por último, los paneles (h-m) muestran una ampliación de los paneles (a-g) en una región en la que se produce una bifurcación.

cuyas derivadas parciales segundas existen y son continuas en el dominio de f, se puede definir la matriz hessiana de la función:

$$\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} \equiv \frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j}. \tag{3.6}$$

Se trata de una matriz cuadrada de dimensión $n \times n$, de manera que

$$\boldsymbol{H}(f)(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_n \partial x_n}. \end{pmatrix} \tag{3.7}$$

Las derivadas segundas pueden entenderse como una medida de la curvatura de la superficie de la función en un determinado punto. Dada una función cuadrática (o una aproximación cuadrática de una función) evaluada en $\boldsymbol{x}$:

- Si $\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} = 0$, la superficie de la función es completamente plana en $\boldsymbol{x}$.
- Si $\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} < 0$, la superficie de la función presenta una curvatura hacia abajo en $\boldsymbol{x}$, siendo más pronunciada cuanto mayor sea la magnitud de la derivada en valor absoluto.
- Si $\frac{\partial^2 f(\boldsymbol{x})}{\partial x_i \partial x_j} > 0$, la superficie de la función presenta una curvatura hacia arriba en $\boldsymbol{x}$, siendo más pronunciada cuanto mayor sea la magnitud de la derivada en valor absoluto.

3.5.2. Los autovalores de la matriz hessiana

Si las derivadas parciales de segundo orden son continuas, los operadores diferenciales son conmutativos, haciendo que la matriz hessiana sea simétrica, tal que $\boldsymbol{H}(f)(\boldsymbol{x})_{i,j} = \boldsymbol{H}(f)(\boldsymbol{x})_{j,i}$. Dado que se trata entonces de una matriz real y simétrica, se puede descomponer la matriz hessiana en un conjunto de autovalores reales y una base ortogonal de autovectores asociados a los mismos.

Cuando se trabaja con múltiples dimensiones, en cada punto de la función existe un valor de la derivada segunda para cada dirección. Para una dirección determinada por un vector unitario $\boldsymbol{d}$, el valor de la derivada segunda puede calcularse como $\boldsymbol{d}^T \boldsymbol{H} \boldsymbol{d}$.

Cuando $\boldsymbol{d}$ es un autovector de $\boldsymbol{H}$, la derivada parcial de segundo orden es igual al autovalor asociado al mismo. Para otras direcciones del vector $\boldsymbol{d}$, la derivada segunda es una media ponderada de todos los autovalores de la matriz hessiana, con pesos entre 0 y 1, siendo los autovalores cuyo autovector asociado presente un menor ángulo con $\boldsymbol{d}$ los que mayor peso reciben.

El estudio de los autovalores de la matriz hessiana toma especial relevancia a la hora de entender los métodos de optimización empleados en *machine learning*. Se ha podido comprobar experimentalmente que durante el entrenamiento de redes neuronales en problemas de clasificación, los autovalores de la matriz hessiana se dividen en dos grupos: la mayoría de los autovalores toman valores cercanos a cero, y representan las direcciones en las que las superficie de la función es prácticamente plana; en cambio, existe un pequeño conjunto de autovalores que toman valores de magnitud elevada, los cuales representan un subespacio con una curvatura más pronunciada y es donde se produce el aprendizaje (Sagun et al., 2017; Sagun et al., 2018).

Estos casos aislados de autovalores de magnitud elevada coinciden en número con la cantidad de clases en las que se dividen los datos de entrenamiento. Además, se ha demostrado que en los métodos de optimización por descenso de gradiente, el sistema evoluciona durante la mayoría del tiempo de entrenamiento en el subespacio definido por los autovectores asociados a dichos autovalores (Gur-Ari et al., 2018). Por esta razón, en la figura 3.5 se puede observar como se han representado únicamente estos tres valores propios, ya que son los que van a definir la curvatura de la superficie por la que se desplaza el sistema durante el entrenamiento.

3.5.3. Evolución temporal de la curvatura

Cada valor de la función de pérdida se corresponde a unos determinados valores de los parámetros $\boldsymbol{W}$. El valor que toma la función en cada punto será mayor cuanto peor clasifiquen estos parámetros los datos de entrenamiento y, por el contrario, menor cuanto mejor los clasifiquen.

Además, en cada valle se va a clasificar mejor unas clases u otras según la región de la superficie en la que se encuentre. Por lo tanto, debido a su naturaleza, los valles de la función de pérdida dinámica van a sufrir variaciones en su profundidad y su anchura según se esté enfatizando una clase u otra (Ruiz-García et al., 2019).

Durante la primera mitad de un periodo T, en la cual se enfatiza el aprendizaje de la clase i, el sistema trata de descender por la superficie de la función hacia un mínimo que permita clasificar correctamente el mayor número de puntos de la clase i. Suponiendo que el valle en el que se encuentra el sistema favorece el aprendizaje de esta clase i, a medida que el valor de Γ_i va incrementando, el valle se vuelve más profundo y se ensancha. Esté último fenómeno se ve reflejado en el hecho de que los autovalores de la Hessiana decrecen. En este caso, por lo tanto, encontramos dos acontecimientos simultáneos:

- La red neuronal se centra en aprender la clase i y, según avanza el aprendizaje, los datos mal clasificados pertenecientes a otras clases contribuyen cada vez menos a la función de pérdida, ya que $\Gamma_i(t) > \Gamma_i(t-1)$ y $\Gamma_{j\neq i}(t) < \Gamma_{j\neq i}(t-1)$. Esto hace que el valle en el que se encuentra descendiendo el sistema se desplace hacia abajo.
- De manera simultanea, y por las mismas razones, el valle también se ensancha. Esto se produce a la vez que otros valles, en los que se clasifican correctamente menos datos de la clase i, se estrechan (y se desplazan hacia arriba).

En la segunda mitad del periodo T, Γ_i decrece según avanza t. Como la superficie de la función está recuperando su topografía original, el valle en el que se encuentra el sistema se eleva y estrecha, lo que implica que los autovalores de la Hessiana crecen.

En esta segunda mitad del periodo se puede observar que:

- La red neuronal enfatiza cada vez menos el aprendizaje de la clase i y los datos mal clasificados pertenecientes a otras clases van cobrando relevancia según se avanza en el aprendizaje, ya que $\Gamma_i(t) < \Gamma_i(t-1)$ y $\Gamma_{j\neq i}(t) > \Gamma_{j\neq i}(t-1)$. Esto hace que el valle en el que se encuentra descendiendo el sistema se desplace hacia arriba.
- De manera simultanea, y por las mismas razones, el valle también se estrecha. Esto se produce a la vez que otros valles, en los que se clasifican mejor los datos de las otras clases, se ensanchan (y se desplazan hacia abajo).

Todo este proceso puede entenderse al observar el panel (f) de la figura 3.5. Los tres autovalores crecen a la vez de que la función de pérdida dinámica decrece durante la primera mitad del periodo. El proceso contrario ocurre durante la segunda mitad.

Llega un momento en el que el sistema comienza a descender por valles que favorecen a las tres clases, por lo que la curvatura de la superficie de dicha región nunca llega a ser tan pronunciada como en los primeros periodos. Además, estos valles son los suficientemente profundos como para lograr altos valores de precisión (en el caso de la figura 3.5, valores de ~ 1).

3.6. El fenómeno de las bifurcaciones

Durante la segunda mitad de algunos periodos T, se produce también un fenómeno destacable a medida que $\Gamma_i \to 1$: aparece una inestabilidad en forma de bifurcación. Además, según avanza t, se producen nuevas inestabilidades, dando lugar a una cascada de bifurcaciones (Ruiz-García et al., 2021).

Un efecto similar fue descrito por Lewkowky *et al.*, en una investigación donde se estudia la dinámica del aprendizaje empleando magnitudes del tasa de aprendizaje elevadas (Lewkowycz et al., 2020).

De la misma forma, la magnitud de la tasa de aprendizaje toma especial relevancia en la aparición de este fenómeno en la función de pérdida dinámica. Cuando la tasa de aprendizaje es lo suficientemente elevada y el valle en el que se encuentra el sistema es lo suficientemente estrecho, el sistema es incapaz de descender con normalidad. En esta situación, el sistema *rebota* en el entorno del mínimo local.

Como se explicó en el apartado anterior, los autovalores de la Hessiana cuantifican la curvatura de la superficie del valle en el que se encuentra el sistema, siendo el máximo autovalor (λ_{max}) el de mayor relevancia. Se ha observado que cuando λ_{max} supera un determinado umbral λ_{max}^{Th}, se produce la inestabilidad. De la misma forma, a medida que los siguientes autovalores sobrepasan el mismo umbral, aparecen nuevas inestabilidades sobre las anteriores formando la cascada de bifurcaciones.

Cabe destacar que en la figura 3.5, se observa como los tres autovalores se bifurcan al comenzar la inestabilidad, pero el segundo y tercer autovalor de la Hessiana no llegan a alcanzar el umbral de λ_{max} antes de que esto se produzca. Generalmente, las inestabilidades desaparecen al terminar el periodo T. Al comenzar el siguiente periodo, la función de pérdida comienza a enfatizar el aprendizaje de otra clase. Una vez que el sistema alcance un valle que favorezca el aprendizaje de esta nueva clase, el valle comenzará a ganar profundidad y anchura, haciendo que el descenso por el mismo sea continuo y suave.

3.6.1. Dependencia del umbral con la tasa de aprendizaje

Suponiendo una función de pérdida cuadrática (o una aproximación cuadrática de una función), la derivada segunda en una determinada dirección permitiría saber cómo de bueno sería el descenso de gradiente si se avanzara en dicha dirección. Una derivada segunda igual a cero indicaría que la superficie de la función es plana en ese punto, por lo que el avance del sistema podría determinarse solamente con el gradiente. Si el gradiente fuese igual a 1 y se toma un paso de tamaño η, la función de pérdida disminuiría en una cantidad igual a η. En cambio, si la derivada segunda es negativa, la superficie presenta una curvatura hacia abajo, por lo que la función de pérdida disminuiría en un valor mayor que η. Para el caso en que la derivada segunda es positiva, la superficie presentaría una curvatura hacia arriba, así que la función de pérdida disminuiría en una cantidad menor que η.

Para el caso de la función de pérdida dinámica, en cada paso t del proceso de optimización el sistema avanza por la superficie de la función según la metodología del descenso de gradiente, donde el sistema sigue el gradiente negativo de la función de pérdida $-\nabla\mathcal{F}$. Una aproximación cuadrática de la función de pérdida dinámica $\mathcal{F}$ mediante el polinomio de Taylor en el entorno de un punto $\boldsymbol{a}$ cualquiera, tomaría la forma

$$\mathcal{F}(\boldsymbol{x}) \sim \mathcal{F}(\boldsymbol{a}) + \nabla\mathcal{F}(\boldsymbol{a})(\boldsymbol{x}-\boldsymbol{a}) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{a})^T \boldsymbol{H}\mathcal{F}(\boldsymbol{a})(\boldsymbol{x}-\boldsymbol{a}), \tag{3.8}$$

donde $\boldsymbol{H}(\mathcal{F})(\boldsymbol{a})$ es la matriz Hessiana evaluada en $\boldsymbol{a}$ (Ruiz-García et al., 2019).

Definiendo una tasa de aprendizaje η tal que $(\boldsymbol{x}-\boldsymbol{a}) \propto \eta$, se puede emplear entonces la ecuación (3.8) para aproximar el valor de la función en $\boldsymbol{x} = \boldsymbol{a} - \eta\nabla\mathcal{F}(\boldsymbol{a})$ (Goodfellow et al., 2016):

$$\mathcal{F}(\boldsymbol{a}-\eta\boldsymbol{g}) \sim \mathcal{F}(\boldsymbol{a}) - \eta\boldsymbol{g}^T\boldsymbol{g} + \frac{1}{2}\eta^2\boldsymbol{g}^T\boldsymbol{H}\mathcal{F}(\boldsymbol{a})\boldsymbol{g}. \tag{3.9}$$

donde se ha simplificado la expresión, de manera que $\boldsymbol{g} = \nabla\mathcal{F}(\boldsymbol{a})$.

La aproximación cuadrática (3.9) da lugar a tres términos: el valor original de la función, el decrecimiento esperado debido a la pendiente de la función (términos de primer orden) y la corrección debido a la curvatura (términos de segundo orden). Cuando los términos de primer y segundo orden en (3.9) son del mismo orden la aproximación de Taylor predice que una tasa de aprendizaje η implicaría que tomar un paso en la dirección $-\nabla\mathcal{F}$ podría hacer al sistema avanzar hacia valores superiores de la función de pérdida, dando lugar a las inestabilidades en forma de bifurcación.

Para que estos dos términos sean del mismo orden, como son proporcionales a η y η^2, respectivamente, se obtiene que el mayor autovalor de la matriz Hessiana, λ_{max}^{Th}, debe escalar como:

$$\lambda_{max}^{Th} \propto \eta^{-1}, \tag{3.10}$$

asumiéndose que $-\nabla\mathcal{F}$ no es perpendicular al autovector asociado a λ_{max}^{Th}. Esta relación pudo ser demostrada experimentalmente por Ruiz *et al.* como se muestra en la figura 3.6.

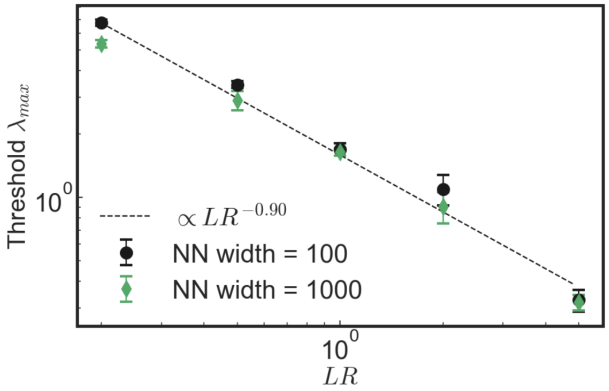

Figura 3.6: Dependencia del umbral de curvatura con la tasa de aprendizaje. El umbral del valor propio más grande de la matriz Hessiana cambia en función de la tasa de aprendizaje para las dos redes empleadas en la figura 3.4, siendo menor cuanto mayor sea la tasa de aprendizaje. Gráfica tomada de (Ruiz-García et al., 2021).

3.6.2. El comportamiento de las bifurcaciones según el Núcleo de la Tangente Neuronal

Jacot *et al.* presentaron en 2020 el Núcleo de la Tangente Neuronal (NTN) como en una nueva *función de núcleo* (en inglés, *kernel function*) (Jacot et al., 2020). Las funciones de núcleo son capaces de operar en espacios de elevada dimensión sin llegar a calcular las coordenadas de los datos de entrada. En vez de ello, realizan una multiplicación matricial de la imagen de los puntos dada una función $g(\boldsymbol{x}_i)$.

En el caso del Núcleo de la Tangente de Neuronal, para unos datos de entrada $x_i \in \mathbb{R}^D$, consiste en la multiplicación matricial de las derivadas parciales de la función de puntuación $f(\boldsymbol{x}_i, \boldsymbol{W})$ con respecto a los parámetros $\boldsymbol{W}$, dando lugar a una matriz de dimensión $[D\text{x}D]$. Se ha demostrado que, para el caso límite en el que la red neuronal sea de ancho infinito, cuando se emplea el NTN ésta se aproxima a un clasificador lineal, lo cual facilita el estudio de la optimización por descenso de gradiente.

A pesar de que la función de pérdida dinámica es una transformación de la función de pérdida de entropía cruzada, el NTN se obtiene a partir de otra función de pérdida: el Error Cuadrático Medio (en inglés, *Mean Squared Error*). Ambas funciones son lo suficientemente similares como para explicar intuitivamente el comportamiento de las bifurcaciones según el NTN.

El error cuadrático medio (ECM) se puede formular como

$$\mathcal{L} = \frac{1}{2N} \sum_{i,k} \left(f_k(\boldsymbol{x}_i, \boldsymbol{W}) - y_{i,k} \right)^2 , \tag{3.11}$$

donde cada dato $\boldsymbol{x}_i$ del conjunto total N pertenece a una de las clases $y_{i,k}$. La salida de la red neuronal, dado un dato de entrenamiento $\boldsymbol{x}_i$ y una matriz de parámetros $\boldsymbol{W}$, es $f_k(\boldsymbol{x}_i, \boldsymbol{W})$, siendo un vector $\in \mathbb{R}^{\mathbb{C}}$, en el que el subíndice k se emplea para la notación de sus términos asociados a cada clase. En este caso de ha formulado el ECM dividiendo por un factor igual a $2N$ en vez de N para obtener una expresión más sencilla al derivar.

Empleando el descenso de gradiente, la evolución del parámetro $\boldsymbol{W} = \{w_p\}$ sería

$$\begin{aligned}\frac{\partial w_p}{\partial t} &= -\eta \frac{\partial \mathcal{L}}{\partial w_p} \\ &= -\frac{\eta}{N} \sum_{i,k} \frac{\partial f_k(\boldsymbol{x}_i, \boldsymbol{W})}{\partial w_p} \left(f_k(\boldsymbol{x}_i, \boldsymbol{W}) - y_{i,k}\right).\end{aligned} \tag{3.12}$$

Desarrollando la evolución temporal de la salida de la red neuronal para un elemento aleatorio del conjunto de datos de entrenamiento $\boldsymbol{x'}$ se obtiene

$$\begin{aligned}&\frac{\partial f_{k'}(\boldsymbol{x}', \boldsymbol{W})}{\partial t} = \sum_p \frac{\partial f_{k'}(\boldsymbol{x}', \boldsymbol{W})}{\partial w_p} \frac{\partial w_p}{\partial t} = \\ &-\frac{\eta}{N} \sum_{i,k,p} \frac{\partial f_{k'}(\boldsymbol{x}', \boldsymbol{W})}{\partial w_p} \frac{\partial f_k(\boldsymbol{x}_i, \boldsymbol{W})}{\partial w_p} \left(f_k(\boldsymbol{x}_i, \boldsymbol{W}) - y_{i,k}\right),\end{aligned} \tag{3.13}$$

donde se definirá el NTN como

$$\Theta_{k',k''}(\boldsymbol{x}', \boldsymbol{x}'') = \sum_p \frac{\partial f_{k'}(\boldsymbol{x}', \boldsymbol{W})}{\partial w_p} \frac{\partial f_{k''}(\boldsymbol{x}'', \boldsymbol{W})}{\partial w_p}, \tag{3.14}$$

siendo $\boldsymbol{x'}$ y $\boldsymbol{x''}$ dos elementos cualesquiera del conjunto de datos de entrenamiento. Si se define la diferencia entre la salida de la red neuronal y la clasificación correcta de cada clase como

$$g_k(\boldsymbol{x}) = \left(f_k(\boldsymbol{x}, \boldsymbol{W}) - y_k\right). \tag{3.15}$$

Combinando las ecuaciones (3.13), (3.14) y (3.15) se obtiene

$$\frac{\partial}{\partial t} g_{k'}(\boldsymbol{x}') = -\frac{\eta}{N} \sum_{i,k} \Theta_{k',k''}(\boldsymbol{x}', \boldsymbol{x}'') g_k(\boldsymbol{x}_i). \tag{3.16}$$

En el límite $\eta \to 0$ se puede diagonalizar el NTN. En este caso, se daría un aprendizaje exponencial de los datos de entrenamiento, a una velocidad mayor en la dirección de los autovectores del NTN asociados a los autovalores máximos λ_{max}.

Reescribiendo la ecuación (3.16) de forma que represente de manera discreta la dinámica del sistema para el caso $\eta \to 0$:

$$g_{k'}^{t+1}(\boldsymbol{x}') = g_{k'}^{t}(\boldsymbol{x}') - \frac{\eta}{N} \sum_{i,k} \Theta_{k',k''}^{t}(\boldsymbol{x}', \boldsymbol{x}'') g_k^t(\boldsymbol{x}_i). \tag{3.17}$$

Si se diagonaliza el NTN como $\Theta_{k',k''}^{t}(x', x'')\tilde{g}_j = \lambda_j \tilde{g}_j$, existen tres posibles regímenes:

- $0 < 1 - \frac{\eta}{N}\lambda_j < 1$. Todos las $\tilde{g}_j$ decrecen de manera exponencial.

- $-1 < 1 - \frac{\eta}{N}\lambda_j < 0$. Al incrementarse el valor de los autovalores del NTN, existe aún una convergencia, aunque $\tilde{g}_j$ cambia de signo en cada paso t.
- $1 - \frac{\eta}{N}\lambda_j < -1$. Una vez que los autovalores cruzan un determinado umbral, el aprendizaje es inestable y la magnitud de $\tilde{g}_j$ diverge a la vez que cambia de signo en cada paso t. Esto hace que aparezca una bifurcación.

Estudiando la dinámica del mayor autovalor λ_{max} del NTN, se observa que éste crece durante el entrenamiento hasta que el sistema se desestabiliza en forma de bifurcación. El valor del λ_{max} entonces decrece hasta la finalización de la cascada de bifurcaciones, para luego comenzar a crecer de nuevo (Ruiz-García et al., 2019).

Es importante destacar que, aunque la función de pérdida dinámica cambia su superficie durante el entrenamiento, el NTN no puede tomar en consideración esta transformación. Esto es debido a que depende exclusivamente de los datos de entrenamiento y los parámetros $\mathbf{W}$. A pesar de ello, se pueden interpretar las inestabilidades en términos del NTN si explicamos su origen con un enfoque diferente: las bifurcaciones no aparecen al cambiar la topografía de la función durante el entrenamiento, sino al cambiar la tasa de aprendizaje *efectiva*. Esta tasa de aprendizaje efectiva es la magnitud del paso a la cual el sistema es capaz de descender por un valle sin desestabilizarse.

Según esta perspectiva, las inestabilidades aparecerían en el NTN como bifurcaciones en las que se cambia el signo en cada paso t tomado. Tras el comienzo de la primera bifurcación, el máximo autovalor λ_{max} decrece al intentar *prevenir* la divergencia, como se puede observar en la figura 3.5.

La tasa de aprendizaje efectiva tiene una dependencia compleja con la topografía de la función de pérdida. Esto queda reflejado en el hecho de que no existe un λ_{max}^{Th} como en el caso del autovalor máximo de la matriz Hessiana: el valor de λ_{max} no es el mismo cada vez que comienza una bifurcación.

4. Aplicación en el error cuadrático medio

4.1. Introducción

Tras comprender en profundidad cómo la función de pérdida de entropía cruzada es capaz de mejorar el rendimiento de una red neuronal, se pueden plantear las siguientes cuestiones: *¿Se pueden convertir otras funciones de pérdida en funciones de pérdida dinámicas? ¿Tendrán lugar los mismos mecanismos de aprendizaje? ¿Aparecerán las bifurcaciones?*

Se propone entonces estudiar la implementación de los factores dinámicos en una nueva función de pérdida: el error cuadrático medio (en inglés, *mean squared error*). Esta función, presentada en la ecuación (3.11) al explicar el núcleo de la tangente neuronal, es una de las funciones de pérdida más populares empleadas en el aprendizaje automático.

El error cuadrático medio (ECM) es una función que suele emplearse en tareas de regresión: toma el error entre un valor estimado y el valor real, lo eleva al cuadrado y calcula su promedio. Sin embargo, es comunmente empleada en tareas de clasificación como función de pérdida, ya que penaliza severamente la diferencia entre la clasificación del modelo y la clasificación real.

Para un conjunto de N datos, en el que cada dato $\boldsymbol{x}_j$ pertenece a la clase y_j, el error cuadrático medio toma la forma

$$\mathcal{L} = \frac{1}{N} \sum_{j \leq N} \sum_{i \leq C} \left(f_i(\boldsymbol{x}_j, \boldsymbol{W}) - \hat{y}_{j,i} \right)^2 , \tag{4.1}$$

donde $f_i(\boldsymbol{x}_j, \boldsymbol{W})$ es la salida de la red neuronal para la clase i. El vector $\hat{\boldsymbol{y}}_j \in \mathbb{R}^C$, siendo C el número de clases, es un vector unitario asociado a cada dato $\boldsymbol{x}_j$, cuyas componentes $\hat{y}_{j,i}$ son nulas para todas las clases excepto para la clase correcta ($i = y_j$), que toma valor 1.

Cada componente de la salida de la red neuronal $f_i(\boldsymbol{x}_j, \boldsymbol{W})$ es comparada con $\hat{y}_{i,j}$; de tal forma queal optimizar esta función de pérdida, de manera gradual las componentes de ambos vectores se irán igualando en valor. De esta forma se alcanzará $\mathcal{L} = 0$ cuando para la clase correcta $f_{y_j}(\boldsymbol{x}_j, \boldsymbol{W}) = 1$ y para el resto de clases $f_{i \neq y_j}(\boldsymbol{x}_j, \boldsymbol{W}) = 0$.

La figura 4.1 muestra un primer análisis del error cuadrático medio como función de pérdida, para el que se ha tratado de aprender los datos en espiral de la figura 3.3. Para construir las gráficas se han empleado dos redes neuronales: una de ancho 100 y otra de ancho 700. Se observa que la primera red, claramente subparametrizada, no alcanza valores de precisión demasiado buenos ($\sim 0,8$). En cambio la red de 700, que puede considerarse sobreparametrizada, alcanza fácilmente precisiones de entrenamiento de ~ 1.

En los próximos apartados se presentará la función de pérdida dinámica de error cuadrático medio, estudiando su aplicación en redes neuronales sencillas, al igual que se hizo con la pérdida de entropía cruzada.

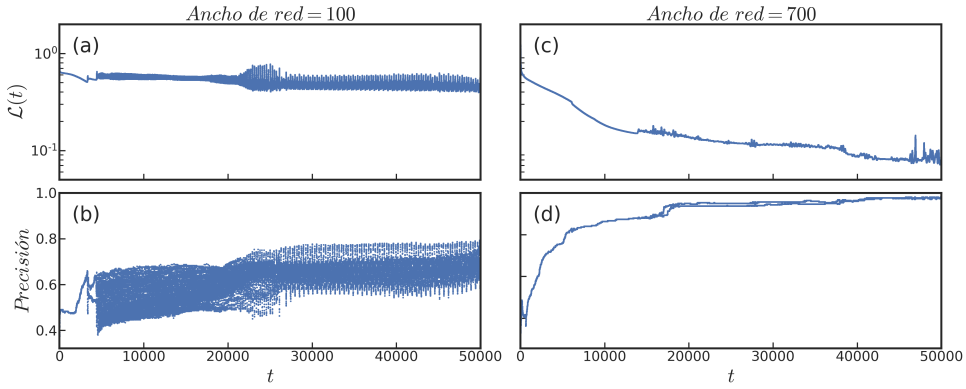

Figura 4.1: Resultados del entrenamiento de una red de ancho igual a 100 (paneles a y b) y otra de 700 (paneles c y d) empleando el error cuadrático medio como función de pérdida, con una tasa de aprendizaje de 0,075. Los paneles (a) y (c) muestran la evolución temporal de la pérdida y los paneles (b) y (d) la precisión de entrenamiento. Cada simulación consta de 50000 pasos.

4.2. Primera versión del ECM dinámico

4.2.1. Formulación

La primera aproximación para convertir el error cuadrático medio en una función de pérdida dinámica será introducir los factores $\Gamma_i(t)$ de manera análoga a como se hizo para el caso de la entropía cruzada, es decir

$$\mathcal{F} = \frac{1}{N}\sum_{j\leq P}\Gamma_i(t)\sum_{i\leq C}\left(f_i(\boldsymbol{x}_j,\,\boldsymbol{W}) - \hat{y}_{j,i}\right)^2. \tag{4.2}$$

En la ecuación 4.2 se asume un conjunto de N datos, en el que cada dato $\boldsymbol{x}_j$ pertenece a la clase y_j; los factores Γ_i empleados fueron definidos en las ecuaciones (3.3), (3.5) y (3.4). Según se expuso previamente, la función de pérdida de ECM en su versión estática es capaz de alcanzar una pérdida de valor $\mathcal{L} = 0$. Al emplear la función de pérdida de entropía cruzada, en cambio, esto no es posible.

Si se presta atención al argumento del logaritmo de la entropía cruzada (véase la ecuación 3.1)

$$\frac{e^{f_{y_j}(\boldsymbol{x}_j,\boldsymbol{W})}}{\sum_i e^{f_i(\boldsymbol{x}_j,\boldsymbol{W})}}, \tag{4.3}$$

donde y_j es la clase correcta, se observa que éste debería tomar un valor igual a 1 para que la pérdida fuese igual a 0. Lo más próximo a este resultado se da cuando

$$e^{f_{y_j}(\boldsymbol{x}_j,\boldsymbol{W})} \to \infty \qquad \text{y/o} \qquad e^{f_{i\neq y_j}(\boldsymbol{x}_j,\boldsymbol{W})} \to 0. \tag{4.4}$$

Esto hace que para alcanzar el valor mínimo de $\mathcal{L}$ éstas funciones de pérdida tomen caminos diferentes: la entropía cruzada dará lugar a una salida de la red neuronal tal que $f(\boldsymbol{x}_j,\,\boldsymbol{W}) \in [-\infty,\infty]$ y la salida empleando el error cuadrático medio será $f_{y_j}(\boldsymbol{x}_j,\,\boldsymbol{W}) \in [0,1]$.

Como se comprobará en los siguientes apartados, esta diferencia hará que al implementar los factores dinámicos, ambas funciones presenten una dinámica de aprendizaje diferente.

4.2.2. Aplicación

Empleando el conjunto de datos de entrenamiento presentado en la figura 3.3, se estudió su implementación en dos redes: una red de una capa oculta de ancho 100 y otra igual de ancho 700.

En la figura 4.2 se presentan los resultados de este estudio. Los paneles (a) y (b) muestran, respectivamente, la precisión de entrenamiento y de evaluación en función de la amplitud (A) y el periodo (T) para la red de ancho igual a 100. Los paneles (c) y (d) muestran lo mismo para el caso de la red de ancho igual a 700. Para cada punto de los paneles (a) y (c) se ha tomado el valor medio de 3 simulaciones, en las que se ha optimizado con descenso de gradiente de lote completo con 50000 pasos y con una tasa de aprendizaje constante e igual a $0,075$. En todas las simulaciones se han parado las oscilaciones en el último periodo (A = 1).

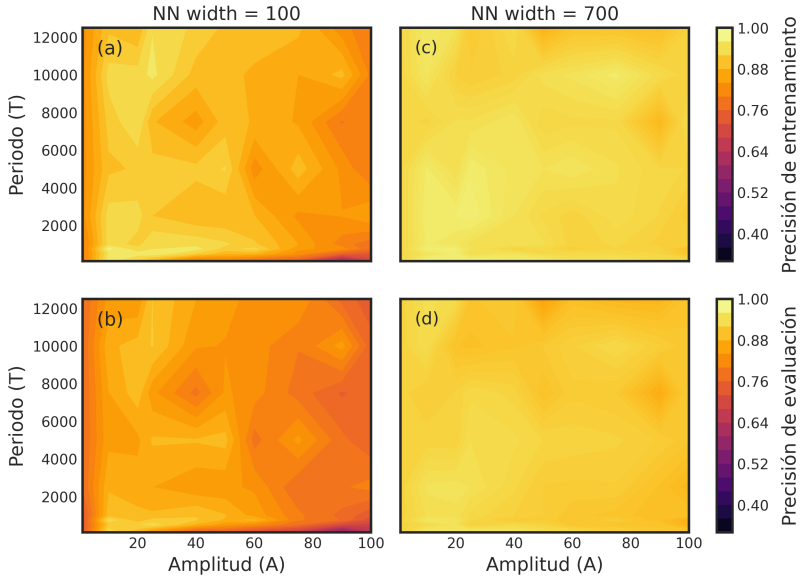

Figura 4.2: Gráficas de contorno de la función de pérdida dinámica de error cuadrático medio, implementada en dos redes neuronales de una capa oculta y con una base de datos en espiral. Los paneles (a) y (b) presentan la precisión de entrenamiento y evaluación, respectivamente, para una red de ancho 100. Los paneles (c) y (d) presentan la precisión de entrenamiento y evaluación, respectivamente, para una red de ancho 700. En ambos casos se ha tomado una tasa de aprendizaje de 0,075 y se ha empleado el descenso de gradiente como método de optimización en simulaciones de 50000 pasos. Cada punto del plano (T,A) de los paneles (a) y (c) es la precisión media de tres simulaciones con diferente semilla de inicialización de los parámetros. Para los paneles (b) y (d) se ha calculado una precisión de evaluación media, empleando los parámetros de entrenamiento de las tres simulaciones, con cinco espirales distintas como conjunto de datos de evaluación.

En el caso de la red subparametrizada de 100 neuronas, igual a la empleada para la figura 4.1, con la función de pérdida convencional ($A = 1$), la red obtiene unos resultados de aprendizaje moderados, lo cual se traduce en una precisión de evaluación de $\sim 0,8$. Sin embargo, al introducir las oscilaciones ($A > 1$) se obtienen muy buenos resultados. Se observa como en varias regiones se alcanzan una precisión de entrenamiento elevada, con valores $> 0,9$. Destaca la región ($10 \lesssim A \lesssim 60, 100 \lesssim T \lesssim 1000$), donde se alcanzan valores de hasta $\sim 0,98$. Esto se traduce en una buena precisión de evaluación en las mismas regiones, con valores de $\sim 0,95$, lo que supone un incremento de $\sim 12\,\%$.

En el caso de la red sobreparametrizada de 700 neuronas, estudiada inicialmente en 4.1, para el caso de la función convencional ($A = 1$), los resultados son ya de por sí muy buenos, con una precisión de entrenamiento y de evaluación $\sim 0,95$. A pesar de ello, al emplear la función de pérdida dinámica se incrementa la precisión de entrenamiento, tomando valores de $\sim$ 0,98. En la evaluación se obtienen resultados igual de buenos, alcanzando valores de $\sim 0,97$.

Cabe destacar que en los cuatro paneles se obtienen buenos valores de precisión en prácticamente toda la región (A, T) estudiada. Además, las elevadas precisiones de evaluación muestran que el modelo no esta especialmente sobreajustado a los datos de entrenamiento.

4.2.3. Dinámica del aprendizaje

Para entender mejor cuáles son los mecanismos de aprendizaje de esta función de pérdida dinámica, se ha elaborado la figura 4.3. Para construirla se ha entrenado la red neuronal de una capa oculta, tomando la espiral como conjunto de datos de entrenamiento, con T = 7500 y A = 40. Para esta simulación larga, de 70000 pasos, se ha definido una tasa de aprendizaje constante $\eta = 0,045$, menor que la empleada en las simulaciones de la figura 4.2. Esto ha sido por el simple motivo de que al representar las gráficas para $\eta = 0,075$, éstas presentaban mucho ruido, dificultando la correcta visualización de los fenómenos presentes.

El panel (a) representa la evolución temporal de los factores $\Gamma_i(t)$. Nótese que su evolución es exactamente igual a la de los factores empleados en la función de pérdida de entropía cruzada.

En el panel (b) puede observarse la evolución temporal de la función de pérdida dinámica $\mathcal{F}(t)$. Al igual que en el caso de la entropía cruzada, la función toma una forma convexa en cada periodo T, representando que durante el mismo se está aprendiendo correctamente la clase que se está enfatizando.

La función de pérdida de error cuadrático medio convencional $\mathcal{L}(t)$ está representada en el panel (c). De nuevo, la función toma una forma cóncava debido a que el énfasis en una de las clases frente al resto, lo cual hace que la función de pérdida convencional tome valores elevados.

La evolución de la precisión de entrenamiento durante el entrenamiento se representa en el panel (e). Se observa que para esta simulación se han alcanzado de nuevo valores elevados de precisión (~ 1).

Al igual que sucedía en el caso de la entropía cruzada, en los paneles (b - e) se puede comprobar que a lo largo del proceso de aprendizaje el sistema se encuentra con situaciones de inestabilidad, lo cual se manifiesta en forma de bifurcaciones. Aquí el fenómeno de las bifurcaciones se presenta de manera completamente diferente con respecto a las bifurcaciones de la entropía cruzada dinámica.

Se trata de unas bifurcaciones muy limpias, con sus ramificaciones bien diferenciadas. Una vez comienza la primera bifurcación, aproximadamente en la segunda mitad del cuarto periodo, ésta se vuelve a cerrar al comenzar el quito periodo. Lo sorprendente es que el quinto periodo ya presenta una bifurcación desde su comienzo. De esto se puede intuir que la bifurcación se mantiene una vez aparece, cerrándose únicamente cuando la función recupera su forma original ($A = 1$).

A pesar de esto, llega un momento en el que aparece cierto ruido. A partir de entonces se pueden hallar momentos de la simulación en los que la bifurcación se cierra y se vuelve a abrir repetidas veces, dentro de un mismo periodo y en intervalos cortos de tiempo.

Estudiando los tres mayores autovalores de la matriz hessiana es posible hacerse una idea de lo que ocurre en la superficie de la función de pérdida durante la minimización. Su evolución temporal queda representada en el panel (f). En este caso, la evolución temporal del autovalor máximo λ_{max} (azul) es completamente diferente a lo que ocurría con la entropía cruzada: durante los primeros periodos crece de manera continua; una vez se desestabiliza el sistema, apareciendo las bifurcaciones, frena su tendencia creciente de forma suave; a partir de ese momento, parece crecer y decrecer en torno a un valor hasta el final de la simulación, con una ligera tendencia creciente. El segundo (naranja) y el tercer (verde) autovalor por debajo de λ_{max} parecen seguir una trayectoria similar.

El umbral λ_{max}^{th} a partir del cual comienzan las bifurcaciones está pintado como una linea roja en la gráfica. Una vez que λ_{max} pasa este umbral, no vuelve a tomar valores inferiores al mismo durante el resto del entrenamiento. También se puede destacar que al pasar el segundo mayor autovalor el umbral, es cuando aparece el ruido en las bifurcaciones. Además, este autovalor tampoco vuelve a tomar valores inferiores al umbral una vez lo pasa.

De estos resultados puede interpretarse que la curvatura de la superficie por la que desciende el sistema no realiza el movimiento peristáltico que se explicó para el caso de la entropía cruzada, en el que el valle realizaba dos movimientos simultáneos: se hacía más profundo y se ensanchaba, para luego elevarse y hacerse más estrecho. En cambio, esta función dinámica realiza al menos uno de los movimientos descritos: el valle por el que desciende el sistema se hace más profundo según se enfatiza la clase i y se eleva a su posición original según el resto de clases toman relevancia. Esto queda plasmado en las dos versiones de la función de pérdida (paneles b y c). Además de este movimiento, el valle en el que se encuentra el sistema se estrecha progresivamente hasta alcanzar una curvatura límite. Esto se observa en el valor creciente de los autovalores hasta llegar al umbral, a partir del cual reducen drásticamente la tendencia creciente.

En el panel (g) se representa el máximo autovalor del Núcleo de la Tangente Neuronal $\lambda_{max,NTN}$, calculado según se presentó en el apartado 3.6.2. El análisis del NTN cobra más sentido al estarse trabajando en ambos casos con una función de pérdida de error cuadrático medio. El autovalor máximo del NTN crece de manera continua, siguiendo una trayectoria similar a la de los autovalores de la hessiana. Al contrario que en el caso de la entropía cruzada, $\lambda_{max,NTN}$ no decrece al aparecer las bifurcaciones. En cambio, mantiene un comportamiento similar a los autovalores de la matriz Hesiana. Se puede entender que la tasa de aprendizaje efectiva no ha cambiado al aparecer la bifurcación, de manera que el sistema es capaz de descender en valles cada vez más estrechos y las bifurcaciones se producen también cuando el NTK alcanza un valor umbral.

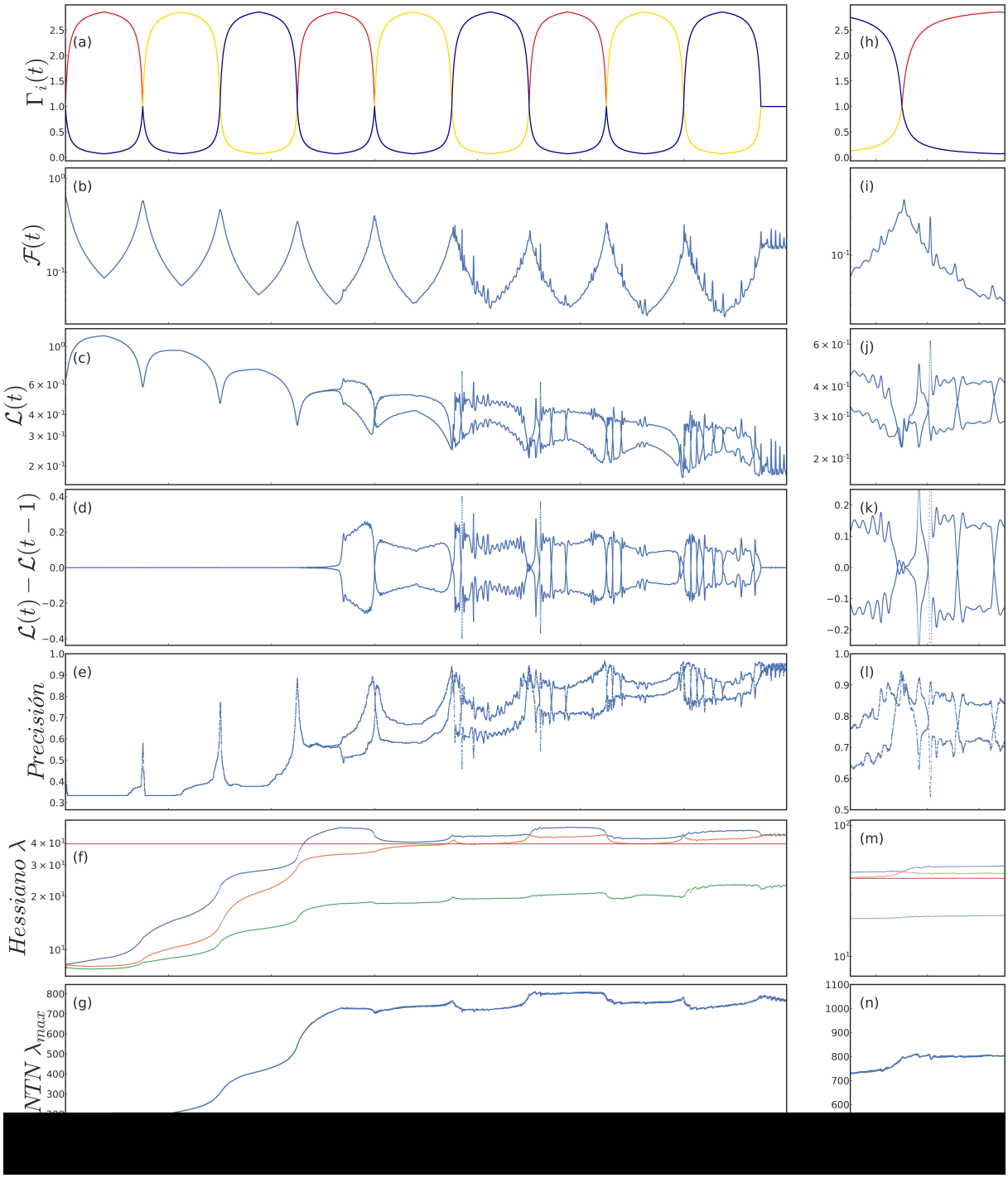

Figura 4.3: Dinámica del aprendizaje al emplear la primera versión de la función de pérdida dinámica de error cuadrático medio. En este caso se empleó una red neuronal de una capa oculta de ancho 100, tomando una amplitud de $A = 40$ y un periodo de $T = 7500$, y con una tasa de aprendizaje igual a 0,045. El conjunto de datos de entrenamiento es en espiral, similar al presentado en la figura 3.3. El panel (a) muestra Γ_i según avanza t, estando cada una de las tres clases representada por un color, al igual que en 3.1. En el panel (b) se representa el valor de la función de pérdida dinámica $\mathcal{F}(t)$. El panel (c) muestra el valor de la función de pérdida convencional $\mathcal{L}(t)$, en la que para todas las clases $\Gamma_i = 1$. En el panel (d) se representa $\mathcal{L}(t) - \mathcal{L}(t-1)$ para exponer de manera clara las inestabilidades. El panel (e) muestra el valor de la precisión durante el entrenamiento. En el panel (f) se representa el valor de los tres mayores autovalores de la Hessiana de la función de pérdida dinámica. En este panel también se ha pintado en rojo el umbral de λ_{max} a partir del cual se desestabiliza el sistema. El panel (g) representa la evolución temporal del autovalor máximo del Núcleo de la Tangente Neuronal (Jacot et al., 2020). Por último, los paneles (h-m) muestran una ampliación de los paneles (a-g) en una región en la que se produce una bifurcación.

Para concluir el análisis de la dinámica del aprendizaje, se ha estudiado la dependencia del umbral λ_{max}^{th} con la tasa de aprendizaje η. Estos resultados se recogen en la figura 4.4. Para la construcción de esta figura se ha tomado el valor medio del umbral en 40 simulaciones de 70000 pasos, con diferentes valores de T y A, y manteniendo la tasa de aprendizaje constante, en una red neuronal de una capa oculta y de ancho 100.

Se ha calculado para las tasas de aprendizaje [0.05, 0.075, 0.1, 0.125], obteniéndose, al igual que en el caso de la entropía cruzada, una relación inversamente proporcional tal que $\lambda_{max}^{Th} \propto \eta^{-1}$. Esto supone que según se aumenta la tasa de aprendizaje, el umbral es menor.

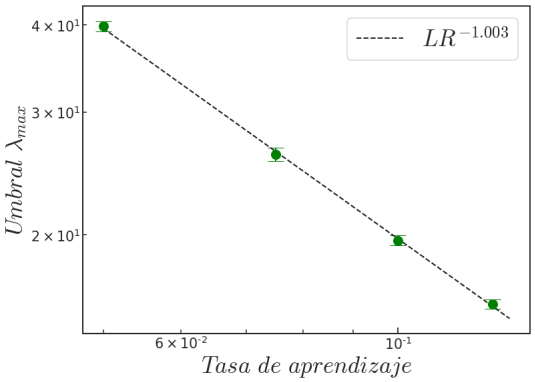

Figura 4.4: Dependencia del umbral de curvatura con la tasa de aprendizaje. El umbral del valor propio más grande de la matriz Hessiana cambia en función de la tasa de aprendizaje para la red de ancho 100 empleada en la figura 4.2. Para cada una de las tasas de aprendizaje se ha calculado un valor medio para el umbral tomando 40 simulaciones de 70.000 pasos con diferentes periodos T y amplitudes A. Cada punto representa la media del valor del umbral con sus respectivos intervalos de confianza del 95 %.

4.3. Segunda versión del ECM dinámico

4.3.1. Formulación

Tras comprobar las diferencias entre la función de pérdida de entropía cruzada y la de error cuadrático medio, se ha trabajado en la implementación de los factores dinámicos en el ECM, de tal forma que la dinámica del aprendizaje sea similar a la que sucede al emplear la entropía cruzada dinámica. Esta segunda versión de la formulación se presenta en la siguiente ecuación:

$$\mathcal{F} = \frac{1}{N} \sum_{j \leq P} \sum_{i \leq C} \left(f_i(\boldsymbol{x}_j, \boldsymbol{W}) - \hat{\Gamma}_i(t) \hat{y}_{j,i} \right)^2 . \tag{4.5}$$

De nuevo, se asume un conjunto de datos N, en el que cada dato $\boldsymbol{x}_i$ pertenece a una clase y_i. En este caso, los factores $\hat{\Gamma}_i$ estan dentro de la función de pérdida convencional, multiplicando a $\hat{\boldsymbol{y}}_j$. Además, no estan normalizados, por lo que quedan definidos según la ecuaciones (3.3) y (3.4). En la figura 4.5 queda representada la evolución temporal de estos nuevos factores.

Tanto en la función de pérdida dinámica de entropía cruzada como en la de error cuadrático medio en su primera versión, los factores $\Gamma_i(t)$ tenían como objetivo que la pérdida total fuese una suma ponderada, en la que se daría un mayor peso a la clase i que se está enfatizando en cada periodo. En este nuevo caso, el enfoque es diferente: al estar los factores $\hat{\Gamma_i}(t)$ multiplicando al vector $\hat{\boldsymbol{y}}_j$, se está forzando al sistema a que se le asigne una puntuación $f_{y_j}(\boldsymbol{x}_j, \boldsymbol{W}) = \hat{\Gamma_i}(t)$ a la clase correcta y al resto de clases $f_{j\neq i}(\boldsymbol{x}_j, \boldsymbol{W}) = 1$. Esta situación hace que el sistema se centre en la clase i, dando lugar a un aprendizaje similar al de las otras dos funciones dinámicas.

De esta forma, se está logrando que $f(\boldsymbol{x}_i, \boldsymbol{W}) \in [1, \hat{\Gamma_i}(t)]$. Esta es una situación intermedia entre la entropía cruzada y la primera versión del ECM.

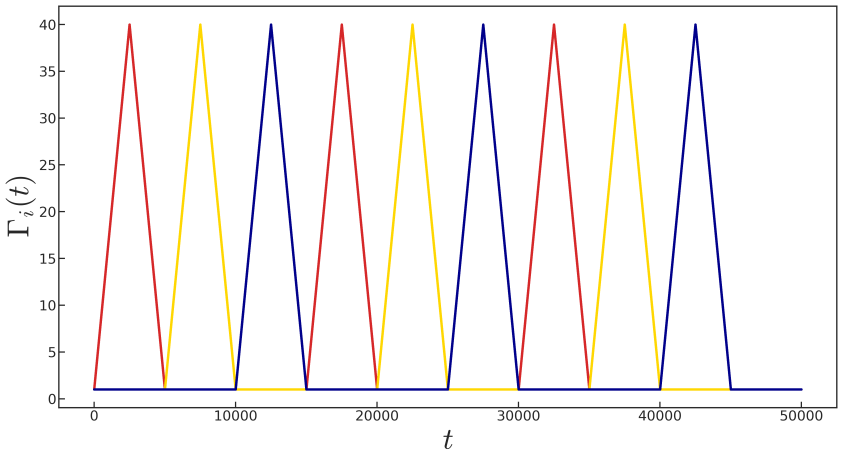

Figura 4.5: Ejemplo de la evolución temporal del valor de los factores asociados a cada clase para la segunda versión de la función de pérdida dinámica de ECM.

4.3.2. Aplicación

Empleando nuevamente conjunto de datos de entrenamiento en espiral de la figura 3.3, se estudió su implementación en una red de una capa oculta de ancho 100 (subparametrizada) y otra igual de ancho 700 (sobreparametrizada), al igual que al estudiar la primera version de la función.

La figura 4.6 muestra las gráficas de contorno obtenidas. Los paneles (a) y (b) muestran, respectivamente, la precisión de entrenamiento y de evaluación en función de la amplitud (A) y el periodo (T) para la red de ancho igual a 100. Los paneles (c) y (d) muestran lo mismo para el caso de la red de ancho igual a 700. Para cada punto de los paneles (a) y (c) se ha tomado el valor medio de 3 simulaciones, en las que se ha optimizado con descenso de gradiente de lote completo con 50.000 pasos y con una tasa de aprendizaje constante e igual a 0.075. En todas las simulaciones se han parado las oscilaciones en el último periodo ($A = 1$).

Para la red de ancho 100, con la función de pérdida convencional ($A = 1$), la red obtiene los mismos resultados de aprendizaje con una precisiones de entrenamiento y evaluación $\sim 0,8$, como era de esperar según los resultados obtenidos en la figura 4.2. Al introducir las oscilaciones ($A > 1$), se obtienen unos resultados notablemente distintos con respecto a la primera versión, pero similares a los obtenidos para el caso de la entropía cruzada.

En primer lugar, se observa una gran mejora en la precisión de entrenamiento con respecto a la función de pérdida estática, pasando de una precisión de $\sim 0,8$ a una de $\sim 0,98$. Destaca la región ($10 \lesssim A \lesssim 40$), donde la precisión es $> 0,9$ para cualquier valor de T.

Evaluando el modelo para el conjunto de datos de evaluación, se observan también muy buenos resultados. Existe una pequeña región en ($A \sim 10, T \sim 5000$), donde se alcanza una precisión de ~ 97, lo que supone de nuevo un $12,25\,\%$ de mejora.

En el caso de la red de ancho 700, la precisión de entrenamiento de $\sim 0,95$ se ve incrementada ligeramente a $\sim 0{,}97$. Esto es en la región ($A \sim 10, 7500 \lesssim T \lesssim 12500$). Además, en la precisión de evaluación existe un pequeño margen de mejora, aumentando la precisión de $\sim 0,95$ a $\sim 0,98$.

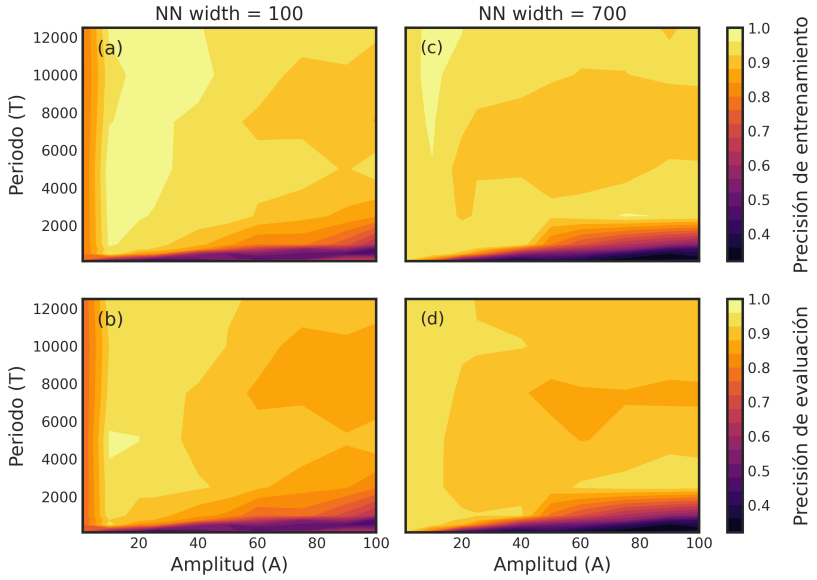

Figura 4.6: Gráficas de contorno de la función de pérdida dinámica de error cuadrático medio (segunda versión), implementada en dos redes neuronales de una capa oculta y con una base de datos en espiral. Los paneles (a) y (b) presentan la precisión de entrenamiento y evaluación, respectivamente, para una red de ancho 100. Los paneles (c) y (d) presentan la precisión de entrenamiento y evaluación, respectivamente, para una red de ancho 700. En ambos casos se ha tomado una tasa de aprendizaje de 0,075 y se ha empleado el descenso de gradiente como método de optimización en simulaciones de 50000 pasos. Cada punto del plano (T,A) de los paneles (a) y (c) es la precisión media de tres simulaciones con diferente semilla de inicialización de los parámetros. Para los paneles (b) y (d) se ha calculado una precisión de evaluación media, empleando los parámetros de entrenamiento de las tres simulaciones, con cinco espirales distintas como conjunto de datos de evaluación.

Al igual que en las gráficas para la entropía cruzada (véase figura 3.4), según se aumenta la amplitud A, la precisión de entrenamiento y de evaluación, en general, disminuyen en ambas redes. A pesar de ello, los resultados que se pueden llegar a obtener son igual de buenos que en la primera versión.

4.3.3. Dinámica del aprendizaje

Nuevamente, se tratará de estudiar los mecanismos de aprendizaje en esta segunda versión del error cuadrático medio dinámico, la cual queda plasmada en las gráficas de la figura 4.3.

Se ha entrenado la red neuronal de una capa oculta tomando la espiral como conjunto de datos de entrenamiento, con $T = 7500$ y $A = 5$. Para esta simulación larga (70000 pasos), se ha definido una tasa de aprendizaje constante $\eta = 0,075$, igual a la empleada en las simulaciones de la figura 4.2.

El panel (a) representa la evolución temporal de los factores $\hat{\Gamma_i}(t)$. Se puede observar en este caso que éstos tienen ahora un crecimiento y decrecimiento lineal, al haberse eliminado la normalización.

En el panel (b) se muestra la evolución temporal de la función de pérdida dinámica $\mathcal{F}(t)$. En este caso la función de pérdida dinámica tiene una evolución más peculiar. Durante la primera mitad del periodo, toma una ligera curvatura cóncava, para luego pasar a ser convexa en la segunda mitad, y finalmente volver a ser cóncava llegando al final del periodo. Además, presenta cierto ruido en algunos tramos. Cabe destacar que a pesar de ello, la red ha aprendido correctamente los datos de entrada ($A \sim 1$), como se puede observar en el panel (e).

Se puede observar en los paneles (c - e), que las bifurcaciones están presentes desde el primer periodo hasta el último, apareciendo siempre llegando al final de cada periodo. Estas bifurcaciones tienen un aspecto completamente distinto al de las bifurcaciones de las otras funciones de pérdida estudiadas, presentando ruido tanto en su apertura como en su cierre.

Analizando los mayores autovalores de la matriz Hessiana, se puede llegar a algunas conclusiones sobre la evolución del sistema y de la curvatura de la función. Los autovalores evolucionan de manera similar a el caso de la entropía cruzada, de forma que según se está enfatizando una clase, los autovalores descienden al ensancharse el valle en el que se encuentra el sistema, para volver a ascender según se acabe el periodo T.

Se observa que estos autovalores nunca pasan un determinado valor. Cuando λ_{max} (azul) alcanza dicho umbral por primera vez, se producen las bifurcaciones, con lo cual este valor coincide con el umbral λ_{max}^{th} (línea roja). Una vez lo alcanza, el autovalor se mantiene constante. Lo mismo ocurre al alcanzar dicho umbral el segundo mayor autovalor (naranja) y, además, aparece cierto ruido en las bifurcaciones cuando esto sucede.

En esta segunda versión de la función de pérdida con error cuadrático medio, se entiende que su superficie sí presenta un movimiento peristáltico similar al caso de la entropía cruzada. Por otra parte, el valor máximo de los autovalores puede entenderse como un límite *real* de curvatura del valle en el que se encuentra el sistema.

En el panel (g) se representa la evolución del autovalor máximo del Núcleo de la Tangente Neuronal. Al igual que sucedía en la función de entropía cruzada dinámica, el autovalor máximo desciende durante las bifurcaciones, intentando controlar la inestabilidad.

Finalmente, al igual que se ha hecho al analizar la dinámica del aprendizaje de otras funciones de pérdida dinámica, se ha estudiado la dependencia del umbral λ_{max}^{th} con respecto a la tasa de aprendizaje η. Los resultados obtenidos se muestran en la figura 4.8.

Para la construcción de esta figura, en cada punto se ha tomado el valor medio del umbral en 40 simulaciones de 70000 pasos, con diferentes valores de T y A, y manteniendo la tasa de aprendizaje constante, en una red neuronal de una capa oculta y de ancho 100.

Se ha calculado para las tasas de aprendizaje [0.05, 0.075, 0.1, 0.125], obteniéndose, al igual que en las otras funciones dinámicas, una relación inversamente proporcional tal que $\lambda_{max}^{Th} \propto \eta^{-1}$. Esto supone que según se aumenta la tasa de aprendizaje, el umbral es menor.

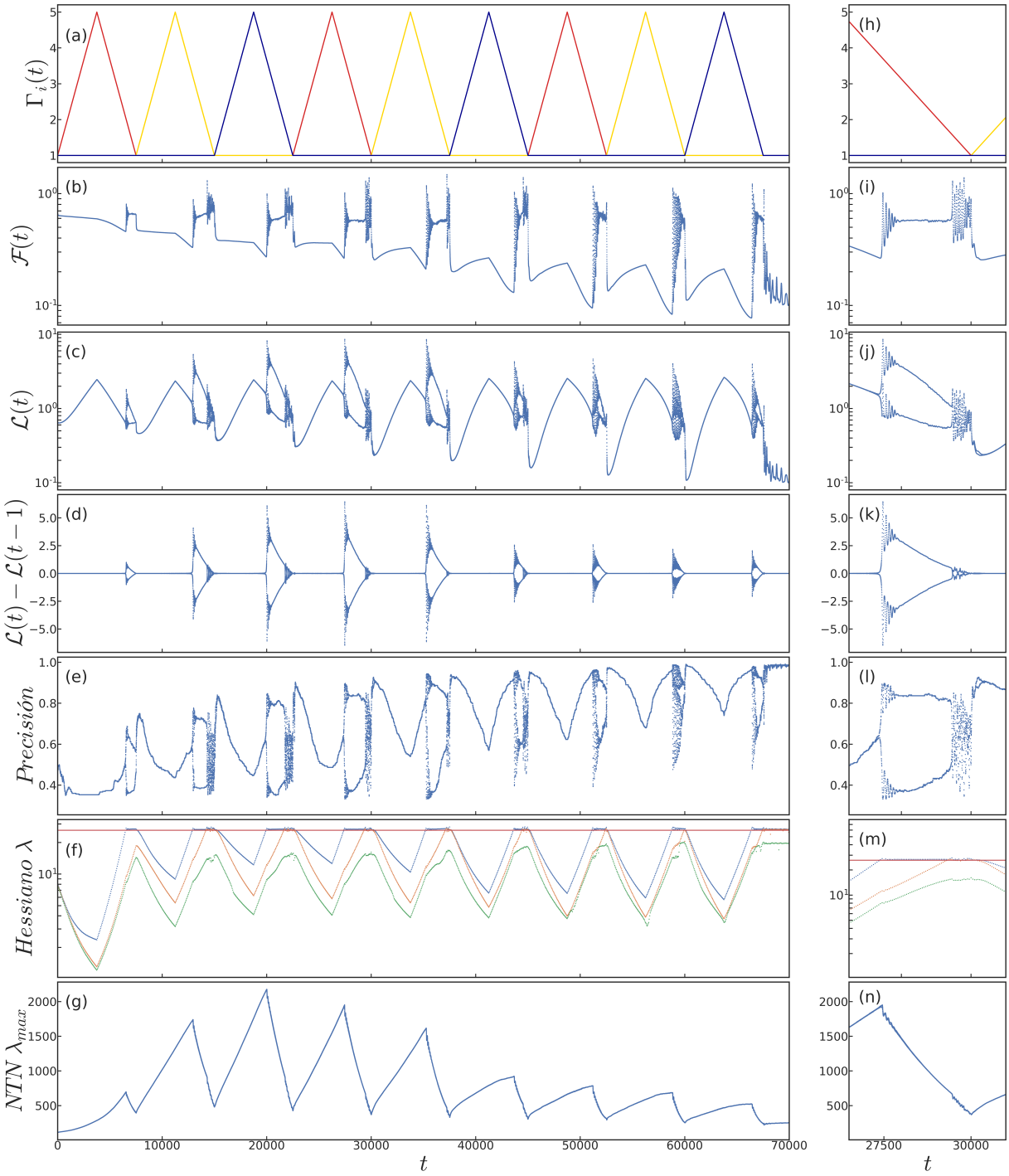

Figura 4.7: Dinámica del aprendizaje al emplear la segunda versión de la función de pérdida dinámica de error cuadrático medio. En este caso se empleó una red neuronal de una capa oculta de ancho 100, tomando una amplitud de $A = 5$ y un periodo de $T = 7500$, y con una tasa de aprendizaje igual a 0,075. El conjunto de datos de entrenamiento es en espiral, similar al presentado en la figura 3.3. El panel (a) muestra Γ_i según avanza t, estando cada una de las tres clases representada por un color, al igual que en 3.1. En el panel (b) se representa el valor de la función de pérdida dinámica $\mathcal{F}(t)$. El panel (c) muestra el valor de la función de pérdida convencional $\mathcal{L}(t)$, en la que para todas las clases $\Gamma_i = 1$. En el panel (d) se representa $\mathcal{L}(t) - \mathcal{L}(t-1)$ para exponer de manera clara las inestabilidades. El panel (e) muestra el valor de la precisión durante el entrenamiento. En el panel (f) se representa el valor de los tres mayores autovalores de la Hessiana de la función de pérdida dinámica. En este panel también se ha pintado en rojo el umbral de λ_{max} a partir del cual se desestabiliza el sistema. El panel (g) representa la evolución temporal del autovalor máximo del Núcleo de la Tangente Neuronal (Jacot et al., 2020). Por último, los paneles (h-m) muestran una ampliación de los paneles (a-g) en una región en la que se produce una bifurcación.

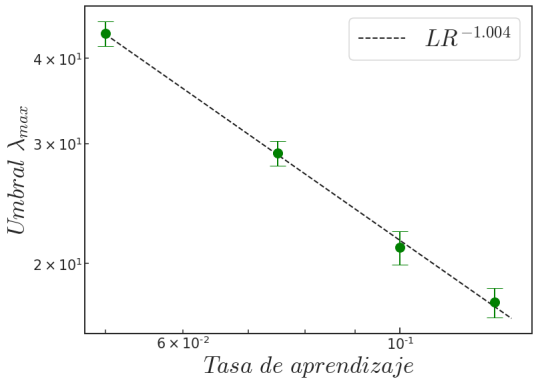

Figura 4.8: Dependencia del umbral de curvatura con la tasa de aprendizaje. El umbral del valor propio más grande del Hessiano cambia en función de la tasa de aprendizaje para las dos redes empleadas en la figura 3.4, siendo menor cuanto mayor sea la tasa de aprendizaje. Para cada una de las tasas de aprendizaje se ha calculado un valor medio para el umbral tomando 40 simulaciones con diferentes periodos T y amplitudes A.

5. Implementación en Python

En esta sección se presentará el código de *Python* empleado para el entrenamiento de una red neuronal sencilla, similar a las que se han descrito a lo largo de este trabajo. Esto es tanto para la función de pérdida dinámica de entropía cruzada, como para las dos versiones de la función de pérdida dinámica de error cuadrático medio.

Se ha empleado el entorno de programación en línea *Google Colaboratory*. Esta herramienta no requiere configuración o descarga de paquetes en el ordenador, ya que se puede programar en las últimas versiones de *Python 3* simplemente con conectarse a los servidores de *Google*. Además, se tiene acceso a GPUs de alto rendimiento, como puede ser la *NVIDIA Tesla P100* o la *GPU NVIDIA T4*, las cuales reducen el tiempo de entrenamiento considerablemente.

5.1. Importaciones

El primer paso es importar los paquetes y módulos necesarios para la ejecución del código. Se emplearán muchos de los paquetes básicos de *Python*, que se encuentran en el primer bloque de importaciones, como puede ser *numpy* o *matplotlib*.

Se trabajará con un paquete desarrollado por *Google* para trabajar con redes neuronales: *JAX*. Se pueden destacar dos funciones de este paquete: la función *jit*, la cual permite acelerar los cálculos mediante GPU, y la función *grad*, la cual calcula el gradiente de cualquier función, facilitando la ejecución de la propagación inversa.

Además se descargarán los módulos *hessian_computation*, *lanczos* y *density*,[2] necesarios para el cálculo de los autovalores de la matriz hessiana.

```
### PAQUETES Y MODULOS ###

import matplotlib
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
import sys
import time
from matplotlib import gridspec
import pickle
import sys
from collections import namedtuple
import os

import jax
from jax import grad, jit
from jax.tree_util import tree_multimap
import jax.numpy as jnp
from jax import random
from jax.config import config
config.update("jax_enable_x64", True)

import hessian_computation as hessian_computation
import lanczos as lanczos
import density as density_lib
```

[2]El código descrito en este apartado, así como el acceso a estos módulos se puede encontrar en https://github.com/miguel-rg/dynamical-loss-functions

5.2. Variables de entrada

El siguiente paso es definir las variables globales que se emplearán a lo largo del código, como las dimensiones de la capa oculta (*NN_width*), el número de clases de los datos de entrenamiento (C) o la tasa de aprendizaje (*learning_rate*).

Todas las variables están explicadas en los comentarios del fragmento de código que se muestra a continuación. La variable lógica *last_period_no_osc*, que cuando es verdadera (valor igual a 1), elimina las oscilaciones en el último periodo del entrenamiento. Mediante la variable *MSE_new_or_old* se indica el tipo de función de pérdida que va a emplearse.

```
### VARIABLES GLOBALES ###

N = 100 # Número de puntos por clase
C = 3  # Número de clases

learning_rate = 1 # Tasa de aprendizaje

T =  2500.          # Tamaño del periodo
total_time = 20000 # Tiempo total de entrenamiento

NN_width = 100 # Ancho de la red

last_period_no_osc = 0 # Puede tomar valores 0 ó 1

MSE_new_or_old = 'cross-entropy'  # Para la pérdida de error cuadrático medio
          ↪ sería 'old' o 'new' según la versión

time_comp_density = 100 # Número de pasos para calcular los autovalores de la
          ↪ Hessiana

order = 10 # Orden del método de Lanczos
```

5.3. Datos de entrada

A continuación, se muestra el código que genera el conjunto de datos de entrenamiento en espiral. Cada brazo de la espiral, correspondiente a una clase, se generará posicionando los puntos a medida que se incrementa un radio y un ángulo de giro. Generalmente es necesario preprocesar los datos de entrenamiento, pero en este caso se han generado de tal manera que ya están centrados y normalizados.

La posición de los datos en el plano queda recogida en la matriz X, correspondiendo cada fila a las coordenadas de cada punto. Además, se define una matriz Y, que determina la clase de cada dato de entrenamiento. Cada fila de Y, asociada a un determinado punto, presenta una componente por clase, siendo todas nulas excepto la clase correcta, que toma valor 1.

```
### DATOS DE ENTRENAMIENTO ###

def make_dataset(points_per_class, classes, revolutions=4):
  np.random.seed(0)

  N = points_per_class
  C = classes
  pi = np.pi

```

```
  X = np.zeros((N*C, 2))
  y = np.zeros((N*C, C))

  for j in range(C):
    ix = range(N*j, N*(j + 1))
    r = np.linspace(0., 1, N) # Radio
    omega = 2*pi/C
    theta_max = revolutions*pi
    t = np.linspace(omega*j, omega*j + theta_max, N) + np.random.randn(N)*0.2 #
        ↪ Ángulo
    X[ix] = np.c_[r*np.cos(t), r*np.sin(t)]
    y[ix, j] = 1

  return jax.device_put(X), jax.device_put

  X, Y = make_dataset(points_per_class=N, classes=C)
```

5.4. Red neuronal

En este fragmento se muestra cómo se emplea la función *stax.serial()* para generar una red neuronal completamente conexa, con una capa oculta y con activación ReLU. Esta red queda definida en la variable *apply_fn*. Además se genera una semilla y se inicializan los parámetros (matriz de pesos y vector de sesgo) con *init_fn*.

```
### RED NEURONAL ###

init_fn, apply_fn, kernel_fn = stax.serial(stax.Dense(NN_width,
          ↪ parameterization='standard'),
                                           stax.Relu(),
                                           stax.Dense(3,
          ↪ parameterization='standard'))

key = random.PRNGKey(0)
_, params = init_fn(key, X.shape)
```

5.5. Núcleo de la tangente neuronal

Mediante la función *nt.empirical_ntk_fn()* se calcula de manera sencilla el Núcleo de la Tangente Neuronal. Posteriormente se calcularán sus autovalores.

```
### NTN Y AUTOVALORES ###

ntk_fn = jit(nt.empirical_ntk_fn(apply_fn)) # NTN

@jit
def ntk_evals(params, X):
  ntk = ntk_fn(X, X, params)
  evals, _ = jnp.linalg.eigh(ntk)
  return evals

ntk_evals(params, X[:50]) # Autovalores del NTN
```

5.6. Funciónes de pérdida

Las funciones de pérdida se van a definir a partir de una serie de funciones, que serán diferentes según se esté tratando de computar la función de pérdida dinámica de entropía cruzada, la función de pérdida dinámica de ECM en su primera versión o la función de pérdida dinámica de ECM en su segunda versión.

- *loss()*: función de pérdida de entropía cruzada convencional.
- *loss_MSE()*: función de pérdida de error cuadrático medio convencional.
- *c_fn()*: función que define los factores dinámicos para la entropía cruzada dinámica.
- *c_fn_MSE_old()*: función que va a definir los factores dinámicos para la primera versión del error cuadrático medio dinámico.
- *c_fn_MSE_new()*: función que va a definir los nuevos factores dinámicos no normalizados para la segunda versión del error cuadrático medio dinámico.
- *weighted_loss()*: función de pérdida dinámica de entropía.
- *weighted_loss_MSE_old()*: función de pérdida dinámica de error cuadrático medio en su primera versión.
- *weighted_loss_MSE_new()*: función de pérdida dinámica de error cuadrático medio en su segunda versión.
- *step_CE()*: permite calcular la actualización de los parámetros durante el descenso de gradiente para función de pérdida dinámica de entropía cruzada.
- *step_MSE_old()*: permite calcular la actualización de los parámetros durante el descenso de gradiente para función de pérdida dinámica de error cuadrático medio en su primera versión.
- *step_MSE_new()*: permite calcular la actualización de los parámetros durante el descenso de gradiente para función de pérdida dinámica de error cuadrático medio en su segunda versión.
- *accuracy()*: calcula la precisión del modelo durante el entrenamiento.

Estas funciones quedan detalladas en el siguiente fragmento:

```
### FUNCIÓN DE PÉRDIDA DINÁMICA CE ###

@jit
def loss(params, X, Y):
  return -jnp.mean(jax.nn.log_softmax(apply_fn(params, X)) * Y) * C

@jit
def loss_MSE(params, X, Y):
  return jnp.mean((apply_fn(params, X) - Y)**2)*C


def c_fn(t,i,w_max):
  slope = 2 * (w_max - 1) / T
  w_main_class = jnp.where(t < T / 2., 1+ t * slope, 2 * w_max - t * slope - 1)
```

```
  res = jnp.ones(C) +  (w_main_class-1) * jnp.eye(C)[i]
  res = res / jnp.sum(res) * C
  return res


def c_fn_MSE_old(t,i,w_max):
  slope = 2 * (w_max - 1) / T
  w_main_class = jnp.where(t < T / 2., 1+ t * slope, 2 * w_max - t * slope - 1)
  res = jnp.ones(C) +  (w_main_class-1) * jnp.eye(C)[i]
  res = res / jnp.sum(res) * C

  mult_1 = res[0]*jnp.ones((N,C))
  mult_2 = res[1]*jnp.ones((N,C))
  mult_3 = res[2]*jnp.ones((N,C))

  mult = jnp.vstack((mult_1,mult_2,mult_3))


def c_fn_MSE_new(t,i,w_max):
  slope = 2 * (w_max - 1) / T
  w_main_class = jnp.where(t < T / 2., 1+ t * slope, 2 * w_max - t * slope - 1)
  res = jnp.ones(C) +  (w_main_class-1) * jnp.eye(C)[i]
  return res, w_main_class


@jit
def weighted_loss(params, X, Y, t, i,w_max):
  w = c_fn(t,i,w_max)
  return -jnp.mean(jax.nn.log_softmax(apply_fn(params, X)) * Y * w) * C


@jit
def weighted_loss_MSE_old(params, X, Y, t, i,w_max):
  w_vec, w_matrix = c_fn_MSE_old(t,i,w_max)
  return jnp.mean(w_matrix*(apply_fn(params, X) - Y)**2)*C


@jit
def weighted_loss_MSE_new(params, X, Y, t, i,w_max):
  w, amp = c_fn_MSE_new(t,i,w_max)
  return jnp.mean((apply_fn(params, X) - w*Y)**2)*C**2/(2+amp**2)


@jit
def step_CE(_, t_and_params_and_c):
  t, params, c, w_max = t_and_params_and_c
  g = grad(weighted_loss)(params, X, Y, t, c, w_max)
  return t + 1, tree_multimap(lambda x, dx: x - learning_rate * dx, params, g),
         ↪ c, w_max


@jit
def step_MSE_old(_, t_and_params_and_c):
  t, params, c, w_max = t_and_params_and_c
  # g = grad(loss_MSE)(params, X, Y)
  g = grad(weighted_loss_MSE_old)(params, X, Y, t, c, w_max)
  return t + 1, tree_multimap(lambda x, dx: x - learning_rate * dx, params, g),
         ↪ c, w_max


@jit
def step_MSE_new(_, t_and_params_and_c):
  t, params, c, w_max = t_and_params_and_c
```

```
  g = grad(weighted_loss_MSE_new)(params, X, Y, t, c, w_max)
  return t + 1, tree_multimap(lambda x, dx: x - learning_rate * dx, params, g),
         ↪ c, w_max


@jit
def accuracy(params, X, Y):
  return jnp.mean(jnp.argmax(apply_fn(params, X), axis=1) == jnp.argmax(Y,
         ↪ axis=1))
```

5.7. Lotes

El fragmento de código que se muestra a continuación tiene como objetivo fragmentar los datos de entrenamiento para realizar un descenso de gradiente por lotes. Como se está trabajando con pocos datos de entrenamiento, se trabaja con lote completo, es decir, que se trabaja con todo el conjunto de datos de entrenamiento.

```
# LOTES #

num_train = X.shape[0]
num_complete_batches, leftover = divmod(num_train, num_train)
num_batches = num_complete_batches + bool(leftover)

print('num_train,num_complete_batches, leftover,num_batches,bool(leftover)',
         ↪ num_train, num_complete_batches, leftover,
 num_batches, bool(leftover))

batch_size = np.shape(X)[0] # El tamaño del lote es igual al tamaño del
         ↪ conjunto de datos de entrenamiento
print('batch_size', batch_size)


def data_stream():
rng = npr.RandomState(0)
while True:
  perm = rng.permutation(num_train)
  for i in range(num_batches):
      batch_idx = perm[i * batch_size:(i + 1) * batch_size]
      yield X[batch_idx], Y[batch_idx]


batches = data_stream()

batches_list = [next(batches) for i in range(num_batches)]


def batches_fn():
for b in batches_list:
  yield b
```

5.8. Autovalores de la matriz hessiana

Los autovalores de la matriz hessiana se obtendrán mediante el algoritmo de Lanczos, un método iterativo para encontrar los autovalores de una matriz a partir del mayor de ellos. La implementación de este algoritmo en *Python* fue presentada en por Ghorbani *et al.* (Ghorbani et al., 2019).

Es necesario hacer uso de los módulos *hessian_computation*, *lanczos* y *density*.

```

### CÁLCULO DE LOS AUTOVALORES DE LA HESSIANA CON ALGORITMO DE LANCZOS EN JAX ###

def compute_density_eigenvalues(params, batches_fn, t, class_w, weight, i):
loss_hvp = lambda params, batch: weighted_loss_MSE_old(params, batch[0], batch[1], t, class_w, weight, T)

hvp, unravel, num_params = hessian_computation.get_hvp_fn(loss_hvp, params, batches_fn)
hvp_cl = lambda v: hvp(params, v) / len(batches_list)  # Match the API required by lanczos_alg

print("num_params: {}".format(num_params))

rng = random.PRNGKey(0)
print("rng: {}".format(rng))
rng, split = random.split(rng)
print('rng:', rng, ', split:', split)
start = time.time()
tridiag, vecs = lanczos.lanczos_alg(hvp_cl, num_params, order, split)
end = time.time()
print("Lanczos time: {}".format(end - start))  # this should be ~order * hvp compute time

print('Time:', t, ', i:', i)
start = time.time()

eig_vals, all_weights = density_lib.tridiag_to_eigv([tridiag])

density, grids = density_lib.eigv_to_density(eig_vals, all_weights, grid_len=10000, sigma_squared=1e-5)

density, grids = density_lib.tridiag_to_density([tridiag], grid_len=10000, sigma_squared=1e-5)
end = time.time()
print("Density computation time: {}".format(end - start))  # this should be ~order * hvp compute time

return density, grids, eig_vals
```

5.9. Gráficas completas

El fragmento de código a continuación define una función que permite obtener las gráficas que muestran la dinámica del aprendizaje. En este caso se representará la evolución en el tiempo de los factores dinámicos $\Gamma_i(t)$, La función de pérdida convencional $\mathcal{L}(t)$, la función de pérdida dinámica $\mathcal{F}(t)$, la precisión de entrenamiento, el mayor autovalor del NTN y los autovalores de la matriz hessiana.

También se va a generar una segunda figura en la que se va a representar la densidad de los autovalores de la matriz Hessiana (Ghorbani et al., 2019).

```
### GRÁFICAS COMPLETAS ###

def plot_all_together(density, grids, NTK_EVALS, W, L, folder_name, ind, L_weighted, A, eigv_list, eigv_plot_times):
```

```
4
5   fig2 = plt.figure(figsize=(50,20))
6
7   markersize_t = 10
8
9   max_eval = [l[-1] for l in NTK_EVALS]
10
11  xlab = lambda name: plt.xlabel(name, fontsize=45)
12  ylab = lambda name: plt.ylabel(name, fontsize=45)
13
14  plt.subplot(2, 3, 1)
15  plt.plot(W,'.', markersize=markersize_t)
16  xlab('$t$')
17  ylab('$w(t)$')
18
19  plt.subplot(2, 3, 2)
20  plt.plot(np.array(L[:-1]) - np.array(L[1:]),'.' )#, markersize=markersize_t)
21  # plt.ylim((-5,5))
22  xlab('$t$')
23  ylab('$L(t) - L(t - 1)$')
24
25  plt.subplot(2, 3, 3)
26  plt.semilogy(L_weighted,'.' )#, markersize=markersize_t)
27  # plt.ylim((0,5))
28  xlab('$t$')
29  ylab('$L_W(t)$')
30
31
32  plt.subplot(2, 3, 4)
33  plt.plot(A,'.' )#, markersize=markersize_t)
34  # plt.ylim((-5,5))
35  xlab('$t$')
36  ylab('$Accuracy$')
37
38  plt.subplot(2, 3, 5)
39  plt.plot(max_eval,'.' )#, markersize=markersize_t)
40  xlab('$t$')
41  ylab('$\\lambda_{max}$')
42
43  plt.subplot(2, 3, 6)
44
45  for iii in range(3):
46    plt.semilogy(eigv_plot_times,eigv_list[:,iii],'.')
47  ylab('$Eigenvalue$')
48  xlab('$t$')
49
50  plt.tight_layout()
51  # fig2.savefig(folder_name+'/complete_sim.pdf')
52  fig2.savefig(folder_name+ '/img_{0}.png'.format(ind))
53
54
55
56  fig3 = plt.figure(figsize=(12,10))
57
58  plt.semilogy(grids, density)
59  plt.ylim(1e-10, 1e2)
60  ylab('$Density$')
61  xlab('$Eigenvalue$')
62  plt.title('time: {}'.format(eigv_plot_times[-1]))
63
64  fig3.savefig(folder_name+ '/density_img_{0}.png'.format(ind))
65
66  plt.show()
```

5.10. Entrenamiento de la red

Este fragmento final realiza el entrenamiento de la red mediante descenso de gradiente de lote completo. Posteriormente, hace una llamada a la función *plot_all_together()* para pintar las gráficas completas del proceso de aprendizaje.

Finalmente, se guardan las variables y los parámetros en un diccionario dentro de un archivo *pickle*.

```
### TRAINING THE NN ###

w_max_list = [1,70,35] # Amplitud máxima de los factores dinámicos

for w_max in w_max_list:

  ind = 0

  # folder_name = '/content/gdrive/My
        ↪ Drive/data_colab_notebooks/NN_Width_{1}_weight_{0}_2'.format(w_max,NN_width)
  # os.mkdir(folder_name)

  print('\n \n \n \n', 'NEW SIMULATION, MAX WEIGHT: ', w_max, '\n \n \n \n')

  L = []
  L_weighted = []
  A = []
  W = []
  NTK_EVALS = []
  eigv_list = []
  eigv_plot_times = []

  c = 0

  _, params = init_fn(key, X.shape)

  w_max_saved = w_max
  told = time.time()
  for i in range(total_time):

    if w_max <1:
      sys.exit('w_max must be >= 1')

    if MSE_new_or_old == 'old':
      t, params, c, w_max = step_MSE_old(0, (t, params, c, w_max))
    elif MSE_new_or_old == 'new':
      t, params, c, w_max = step_MSE_new(0, (t, params, c, w_max))
    elif MSE_new_or_old =='cross_entropy':
      t, params, c, w_max = step_CE(0, (t, params, c, w_max))

    if t > T:

      tnew = time.time()
      print('Period ',int(i%T) ,' finished. Time steps: ',t,' Computation time:
          ↪ ',tnew-told)
      told = tnew

      t = 0
      c = (c + 1) % 3

      if last_period_no_osc:
        if i>total_time-T:
```

```
51            print('last period without oscillations')
52            w_max = 1
53
54      L += [loss_MSE(params, X, Y)]  # Se calcula la función de pérdida
            ↪ convencional
55      A += [accuracy(params, X, Y)]  # Se calcula la precisión del modelo
56
57      # Se calcula la pérdida dinámica y se actualizan los parámetros
58
59      if MSE_new_or_old == 'old':
60        L_weighted += [weighted_loss_MSE_old(params, X, Y, t, c,w_max)]
61        W += [c_fn_MSE_old(t,c,w_max)[0]]
62      elif MSE_new_or_old == 'new':
63        L_weighted += [weighted_loss_MSE_new(params, X, Y, t, c,w_max)]
64        W += [c_fn_MSE_new(t,c,w_max)[0]]
65      elif MSE_new_or_old =='cross_entropy':
66        L_weighted += [weighted_loss(params, X, Y, t, c,w_max)]
67        W += [c_fn(t,c,w_max)]
68
69      NTK_EVALS += [ntk_evals(params, X[:50])]
70
71      if i%time_comp_density == 0:
72
73        print('\n\n Plotting, time: ', i)
74
75        density, grids, eigvs =
            ↪ compute_density_eigenvalues(params,batches_fn,t,c,w_max, i)
76
77        eigv_list += [[eigvs[0][-1],eigvs[0][-2],eigvs[0][-3]]]
78        eigv_list_a = np.array(eigv_list)  # Se guardan los autovales de la
            ↪ Hessiana
79        eigv_plot_times += [i]
80
81        # Se pintan las gráficas completas
82
83        plot_all_together(density, grids, NTK_EVALS, W, L, folder_name, ind,
            ↪ L_weighted, A, eigv_list_a, eigv_plot_times)
84        ind = ind + 1
85
86
87        file_name =  'T_{0}_w_max_{1}_NN_width_{2}.pkl'.format(T,w_max,NN_width)
88
89        dictionary_data =  {'learning_rate': learning_rate,
90                            'grids': grids,
91                            'eigv_plot_times': eigv_plot_times,
92                            'NTK_EVALS': NTK_EVALS,
93                            'density': density,
94                            'eigv_list_a': eigv_list_a,
95                            'NN_width': NN_width,
96                            'last_period_no_osc': last_period_no_osc,
97                            'start_osc_from_the_middle':
            ↪ start_osc_from_the_middle,
98                            'T': T,
99                            'total_time': total_time,
100                           'w_max': w_max_saved,
101                           'loss_function_type': MSE_new_or_old,
102                           'L': L,
103                           'L_weighted': L_weighted,
104                           'W': W,
105                           'A': A,
106                           # 'num_periods': num_periods,
107                           'time_comp_density': time_comp_density,
108                           'Lanczos_order': order
```

```
109                                 }
110
111
112         a_file = open(file_name, "wb")
113         pickle.dump(dictionary_data, a_file)
114         a_file.close()
```

6. Planificación y presupuesto

6.1. Planificación

En este apartado se presenta la *Estructura de Descomposición del Proyecto* (EDP), en la figura 6.1, que muestra de manera organizada cada unas de las tareas llevadas a cabo a lo largo del trabajo. Además, en la figura 6.2 se incluye un *diagrama de Gantt*, que muestra gráficamente la secuencia temporal entre cada una de las tareas.

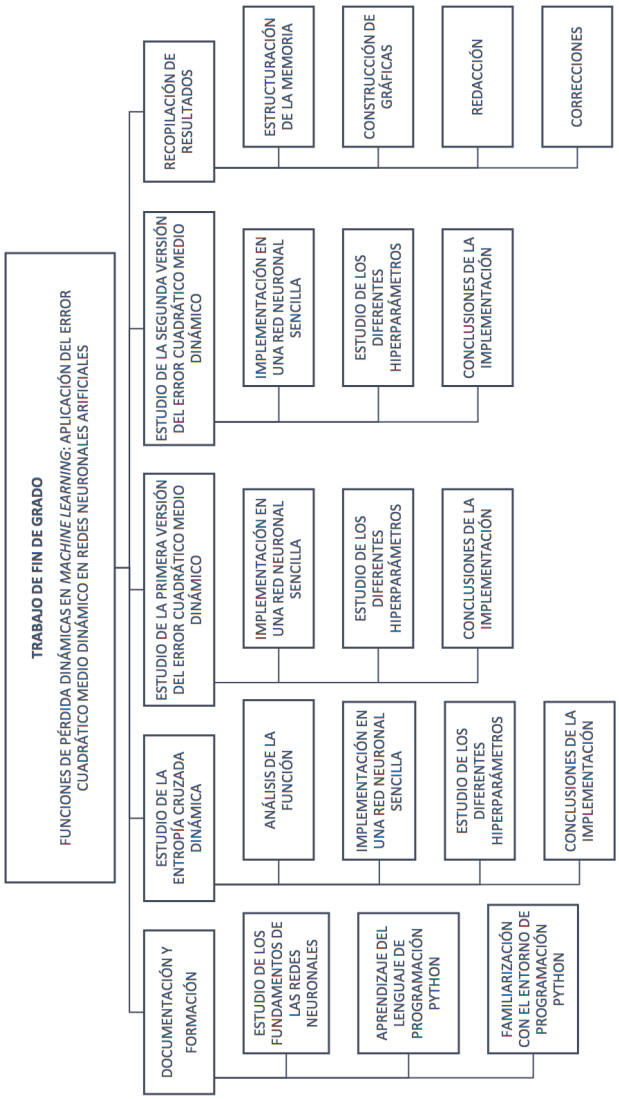

Figura 6.1: EDP del Trabajo de Fin de Grado

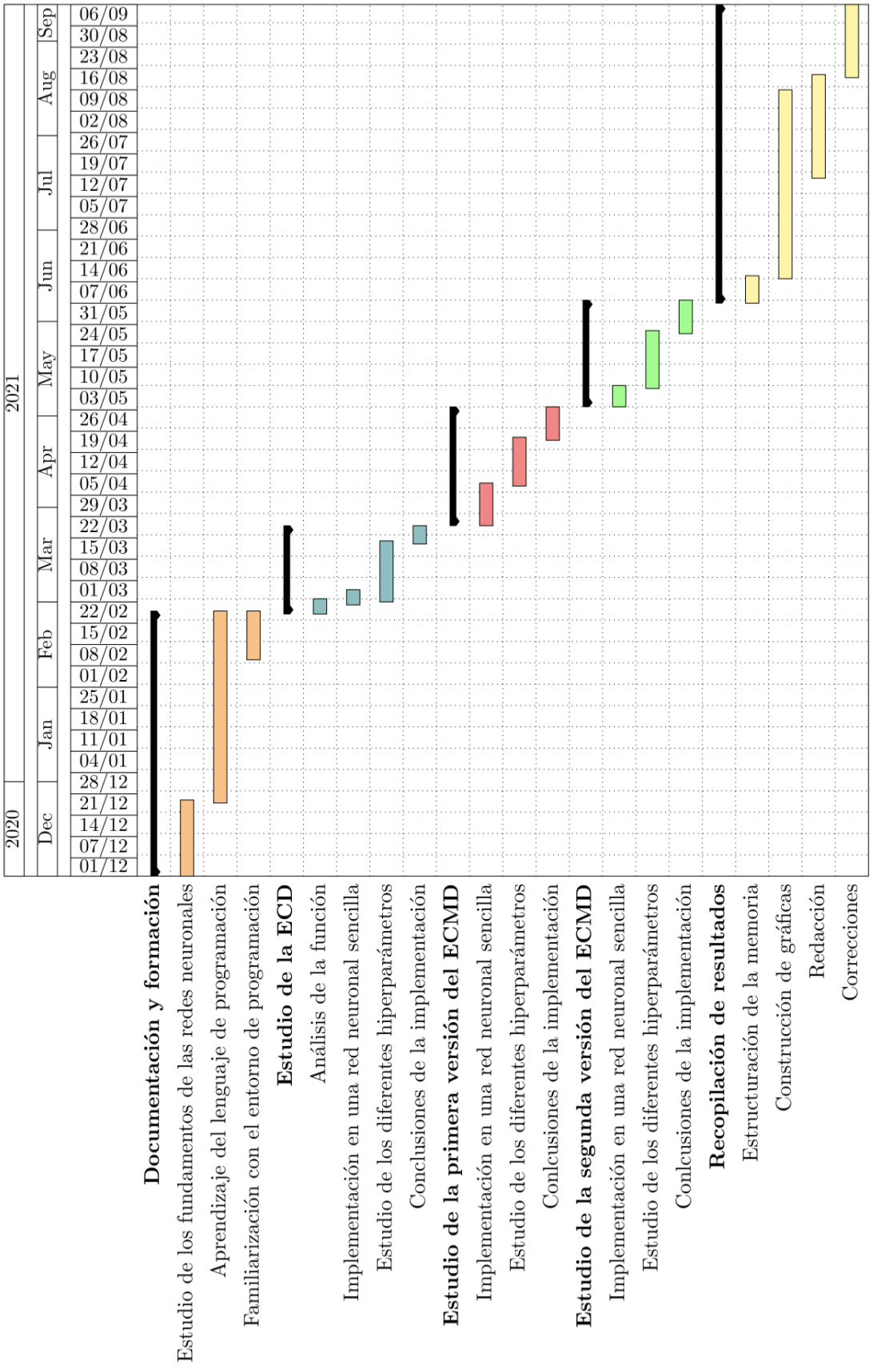

Figura 6.2: Diagrama de Gantt del Trabajo de Fin de Grado

6.2. Presupuesto

Cualquier Trabajos de Fin de Grado, independientemente de su tipología o naturaleza, utilizan unos determinados recursos para su consecución y por lo tanto, requieren de un cierto presupuesto.

Para este proyecto de investigación, el principal factor que entra en juego para su desarrollo es el factor humano. Se ha asumido que el alumno se encuentra realizando el TFG bajo un contrato de colaboración como becario en el *Departamento de Matemáticas Aplicadas a la Ingeniería Industrial de la Escuela Técnica Superior de Ingenieros Industriales*. Se asumirá un contrato de 300 horas de trabajo, por el que se dotará al alumno de un total de 1776€.[3] Esto se traduce en un contrato de 5,96€ por hora, al distribuirse el trabajo a lo largo de 8 meses.

Para la implementación de los algoritmos se hace uso del lenguaje *Python*. A pesar de ser un lenguaje gratuito, debido a las altas horas de entrenamiento es necesario hacer uso deun entorno de programación profesional conocido como *Google Colaboratory*. Se hara usó de la versión *Pro*, de pago, para acelerar los tiempos de simulación. Esta versión tiene un coste de 10€ mensuales.

Por lo tanto, asumiendo que en condición de becario se esta excento de reducciones fiscales y que el precio de *Google Colaboratory Pro* tiene incluidos los impuestos, se presenta en la tabla 6.1 un presupuesto total de 2726€.

CONCEPTO	VALOR	DURACIÓN	TOTAL
Contrato de becario	5,92 €/hora	300 horas	1776 €
Google Colaboratory	10 €/mes	8 meses	950 €
TOTAL			2726 €

Tabla 6.1: Presupuesto del Trabajo de Fin de Grado

[3]Presupuesto tomado según una beca de colaboración propuesta para el curso 20/21. (`https://www.etsii.upm.es/la_escuela/doc/Guia_TFG_TFM_2019.pdf`)

7. Valoración de impactos

La eficiencia de las redes neuronales es un asunto que preocupa a la comunidad científica. En un artículo de 2019 dirigido por Roy Schwartz, titulado *Green AI*, se discute la eficacia de las grandes redes neuronales. A pesar de que su desarrollo ha supuesto grandes avances en el ámbito del aprendizaje automático, logrando modelos con precisiones sin precedentes, se ignora por completo el impacto ambiental, social y económico asociado al mismo (Schwartz et al., 2019).

En un estudio de 2019, dirigido por Emma Strubel, se mostró cómo la elevada precisión de las redes neuronales profundas generalmente requiere una ingente cantidad de recursos computacionales, lo que supone un consumo energético extremadamente alto. De esta forma, desarrollar y entrenar una red neuronal profunda tiene un elevado coste económico, debido al consumo eléctrico y al *hardware*, así como un elevado coste ambiental a causa de la huella de carbono generada al abastecer estos super-algoritmos (Strubell et al., 2019).

En mayo de 2020, la compañía *OpenAI* presentó el mayor modelo de aprendizaje automático no supervisado de la historia: el GPT-3, un modelo de procesamiento de lenguaje natural capaz de realizar una amplia variedad de tareas, que van desde la traducción hasta la resolución de exámenes, para lo cual emplea 175 mil millones de parámetros. Según unos cálculos aproximados obtenidos en un artículo de la revista *Forbes*, el CPT-3 estuvo aproximadamente seis meses entrenando más de 4700 versiones del modelo, con 500 mil millones de palabras como datos de entrenamiento: esto supuso una huella de carbono de 35 toneladas de CO_2, más de lo que puede producir un una persona adulta en 2 años (Toews, 2020).

Uno de los principales problemas de estas redes neuronales masivas es el tiempo que se tarda en ajustar los hiperparámetros de la red y en aprender la enorme cantidad de datos de entrenamiento que necesitan estas redes para generar los modelos. La implementación de las funciones de pérdida dinámicas presentadas en este Trabajo de Fin de Grado podrían suponer una mejora en la eficiencia de las redes neuronales. Al mejorar los resultados de las funciones de pérdida convencionales, es posible emplear arquitecturas más sencillas o que requieran menos tiempo de entrenamiento para aprender correctamente los datos de entrenamiento. De esta forma, se podría lograr reducir el impacto asociado al uso de esta tecnología.

8. Conclusiones

El objetivo general de este Trabajo de Fin de Grado ha sido estudiar la implementación de unos factores dinámicos en el error cuadrático medio, con el fin de emplearlo como función de pérdida en una red neuronal artificial para tareas de clasificación supervisadas. Los resultados de esta investigación continúan estudios previos presentados por el Dr. Ruiz García *el at.* en *Tilting the playing field: Dynamical loss functions for machine learning* (Ruiz-García et al., 2021), lo que supone una contribución académica en el ámbito de las redes neuronales artificiales y al aprendizaje automático en general.

Se han presentado dos formas distintas de formular la que se ha denominado *función de pérdida dinámica de error cuadrático medio*. Cada una de ellas presenta comportamientos diferentes durante el entrenamiento de la red, tanto con respecto a la función de pérdida dinámica de entropía cruzada como entre sí. Sin embargo, se han obtenido resultados prometedores, los cuales refuerzan la idea de que la implementación de una función de pérdida dinámica en redes neuronales logra mejorar el proceso de aprendizaje.

A continuación se muestra de manera detallada las conclusiones que surgen al comparar las funciones presentadas en este trabajo:

- Primera versión del error cuadrático medio dinámico

En la primera versión de la función de pérdida dinámica de error cuadrático medio se han introducido los factores dinámicos de manera análoga a como se implementaron en el caso de la entropía cruzada. Paradójicamente, ambas funciones presentan una dinámica de aprendizaje muy diferente. Esta diferencia de comportamiento se ha asociado a la disparidad entre los valores de salida de la red que requiere cada función para alcanzar sus valores mínimos.

A pesar de ello, se ha demostrado que esta función es igual de válida a la hora de mejorar el aprendizaje. En una red subparametrizada, que no era capaz de aprender, se ha logrado alcanzar valores de precisión de evaluación cercana a 1, lo que supone un incremento de $\sim 12\,\%$ con respecto a los resultados obtenidos en el caso de la misma función de pérdida en su versión estática. En el caso de una red sobreparametrizada, que sí era capaz de aprender correctamente, se ha incrementado ligeramente la precisión de evaluación, lo que se traduce en una mejora del comportamiento del modelo ante nuevos datos de entrada.

En el caso de la función de pérdida dinámica de entropía cruzada, las oscilaciones impuestas en la superficie de la función suponen un movimiento peristáltico del valle en el que se encuentra descendiendo el sistema, en el que en cada periodo el valle se ensancha y profundiza, para luego elevarse y estrecharse. En el caso de la nueva función de pérdida dinámica de error cuadrático medio, se ha podido comprobar que el valle en el que se encuentra el sistema se hace más profundo y se eleva en cada periodo de la misma forma, pero el movimiento de ensanchamiento/estrechamiento se convierte en un proceso progresivo de estrechamiento, durante todo el entrenamiento y hasta un valor límite.

Durante el entrenamiento de la red neuronal, en el caso de la función de pérdida dinámica de entropía cruzada surgen a lo largo del proceso una serie de situaciones de inestabilidad, que se manifestan como cascadas de bifurcaciones. Estas bifurcaciones están asociadas a unos niveles máximos de curvatura de la superficie de la función de pérdida, por lo que aparecen cuando el valle en el que se encuentra el sistema es demasiado estrecho, que suele suceder al final de cada periodo de oscilación. En el caso de la función de pérdida dinámica de error cuadrático medio también aparecen bifurcaciones al pasar un umbral de curvatura, pero al encontrarse en un proceso de estrechamiento constante, las bifurcaciones no convergen una vez han aparecido, permaneciendo abiertas durante el resto del entrenamiento.

Realizando un estudio de los autovalores de la matriz Hessiana de la función de pérdida dinámica de error cuadrático medio se ha podido comprobar que el umbral de curvatura a partir del cual se desestabiliza el aprendizaje es inversamente proporcional a la tasa de aprendizaje, igual que se demostró para el caso de la entropía cruzaada dinámica. Esta observación está en total acuerdo con la predicción teórica que se desarrolló mediante la aproximación por el polinomio de Taylor en el entorno del punto en el que se encuentra el sistema.

- Segunda versión del error cuadrático medio dinámico

En la segunda versión de la función de pérdida dinámica de error cuadrático medio, se han introducido los factores dinámicos de manera diferente a como se ha realizado en las otras dos funciones. Esta implementación presenta una dinámica de aprendizaje muy parecida a la presente en la entropía cruzada dinámica. Esta similitud de comportamiento se ha asociado a que los valores de salida de la red que requiere esta función para alcanzar sus valores mínimos se asemejan un poco más a los valores requeridos para el caso de la entropía cruzada.

Nuevamente, se ha demostrado cómo esta función permite mejorar el aprendizaje de la red neuronal. En una red subparametrizada, que no era capaz de aprender, ha logrado alcanzar, de nuevo, valores de precisión de evaluación cercanos a 1, suponiendo también un incremento de $\sim 12\,\%$ con respecto a los resultados obtenidos en el caso de la misma función de pérdida en su versión estática. En el caso de una red sobreparametrizada, que sí era capaz de aprender correctamente, se ha incrementado ligeramente la precisión de evaluación, lo que se traduce en una mejora del comportamiento del modelo ante nuevos datos de entrada, al igual que ocurría en la primera versión.

Al emplear esta versión de la función de pérdida dinámica de error cuadrático medio se ha podido comprobar que el valle en el que se encuentra el sistema presenta, al igual que en el caso de la entropía cruzada dinámica, un movimiento peristáltico, en el que en cada periodo el valle se ensancha y profundiza, para luego elevarse y estrecharse. De manera similar a su primera versión, el estrechamiento sucede hasta un valor límite de curvatura.

En esta segunda versión de la función de pérdida dinámica de error cuadrático medio también aparecen situaciones de inestabilidad, que se manifestan como una bifurcación en cada periodo. Estas bifurcaciones están asociadas a unos niveles máximos de curvatura de la superficie de la función de pérdida, como en las otras dos funciones, que aparecen cuando el valle en el que se encuentra el sistema es demasiado estrecho. Debido a la dinámica de la superficie de la función, esto suele suceder al final de cada periodo de oscilación, como ocurría en la entropía cruzada dinámica.

Realizando nuevamente un estudio de los autovalores de la matriz Hessiana de esta versión de la función de pérdida dinámica de error cuadrático medio, se ha podido comprobar que el umbral de curvatura a partir del cual se desestabiliza el aprendizaje es inversamente proporcional a la tasa de aprendizaje, igual que se demostró para el caso de las otras funciones. De igual manera, esta observación está en total acuerdo con la predicción teórica que se desarrolló mediante la aproximación por el polinomio de Taylor en el entorno del punto en el que se encuentra el sistema.

- Comparativa de la primera y la segunda versión del error cuadrático medio dinámico

Se ha demostrado cómo cualquiera de las dos versiones de la función de pérdida de error cuadrático medio es capaz de mejorar los resultados de aprendizaje con respecto a la versión estática del error cuadrático medio. A pesar de ello, la primera versión de la función presenta buenos resultados para prácticamente todas las combinaciones de periodo y amplitud de las oscilaciones. En cambio, la segunda versión es más selectiva en cuanto a la amplitud, mostrando peores resultados según se aumente. Además, la segunda versión trabaja notablemente peor para valores bajos de periodo, en los que la frecuencia de las oscilaciones es mayor. A priori, esto puede indicar que la primera versión es más fácil de implementar, ya que no requiere un estudio previo de los valores óptimos de amplitud y periodo.

Por otro lado, la diferencia en la dinámica de aprendizaje entre ambas versiones es notable. De los resultados obtenidos, se puede pensar que el movimiento de ensanchamiento/estrechamiento del valle en el que se encuentra el sistema no juega un papel tan importante en el aprendizaje como la profundización/elevación del mismo. Dicho esto, aún hay camino por delante de cara a determinar el verdadero motivo que hace a la función de pérdida dinámica tan efectiva.

Por último, se puede concluir que, aunque las funciones de pérdida dinámicas obtienen mejores resultados, es necesario conocer en profundidad la función de pérdida en la que se quiera implementar.

8.1. Líneas futuras de desarrollo

La evolución natural de los estudios sobre la función de pérdida dinámica de error cuadrático medio es emplearla en una red neuronal convolucional. Al igual que se hizo para la entropía cruzada dinámica, sería interesante conocer el rendimiento de esta función en una red como la *Myrtle5*, empleando un conjunto de datos más realista que la espiral de puntos, como puede ser la base de datos *CIFAR-10*.

El siguiente paso evidente será estudiar la implementación de los factores dinámicos en otras funciones de pérdida. Esto es necesario para poder demostrar si la función de pérdida dinámica realmente es capaz de mejorar los resultados de aprendizaje de manera general.

Paralelamente, es importante seguir indagando en los mecanismos de aprendizaje inherentes en las funciones de pérdida dinámicas, y de esta forma ajustar de manera óptima su implementación para cada caso.

Índice de figuras

Bibliografía

Arous, Gérard Ben et al. (nov. de 2019). «The Landscape of the Spiked Tensor Model». English (US). En: *Communications on Pure and Applied Mathematics* 72.11. Publisher Copyright: © 2019 Wiley Periodicals, Inc., págs. 2282-2330. ISSN: 0010-3640. DOI: 10.1002/cpa.21861.

Bengio, Yoshua et al. (2009). «Curriculum Learning». En: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, págs. 41-48. ISBN: 9781605585161. DOI: 10.1145/1553374.1553380. URL: https://doi.org/10.1145/1553374.1553380.

Cea, Tommaso et al. (2019). «Large-scale critical behavior of the rippling phase transition for graphene membranes». En: *arXiv preprint arXiv:1911.10536*.

Choromanska, Anna et al. (sep. de 2015). «The Loss Surfaces of Multilayer Networks». En: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Guy Lebanon y S. V. N. Vishwanathan. Vol. 38. Proceedings of Machine Learning Research. San Diego, California, USA: PMLR, págs. 192-204. URL: https://proceedings.mlr.press/v38/choromanska15.html.

Dabbura, Imad (diciembre de 2017). *Gradient Descent Algorithm and Its Variants*. Ed. por Towards Data Science. [Online; posted 21-diciembre-2017]. URL: https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3.

Ghorbani, Behrooz, Krishnan, Shankar y Xiao, Ying (2019). *An Investigation into Neural Net Optimization via Hessian Eigenvalue Density*. arXiv: 1901.10159 [cs.LG].

Glorot, Xavier y Bengio, Yoshua (13–15 May de 2010). «Understanding the difficulty of training deep feedforward neural networks». En: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. por Yee Whye Teh y Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, págs. 249-256. URL: https://proceedings.mlr.press/v9/glorot10a.html.

Goodfellow, Ian, Bengio, Yoshua y Courville, Aaron (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press. Cap. 4, págs. 80-90.

Gur-Ari, Guy, Roberts, Daniel A. y Dyer, Ethan (2018). *Gradient Descent Happens in a Tiny Subspace*. arXiv: 1812.04754 [cs.LG].

He, Kaiming et al. (2016). «Deep Residual Learning for Image Recognition». En: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, págs. 770-778. DOI: 10.1109/CVPR.2016.90.

Jacot, Arthur, Gabriel, Franck y Hongler, Clément (2020). *Neural Tangent Kernel: Convergence and Generalization in Neural Networks*. arXiv: 1806.07572 [cs.LG].

Karpathy, A. (2020). *Convolutional neural networks for visual recognition*. URL: https://cs231n.github.io.

Krizhevsky, Alex (2009). «Learning Multiple Layers of Features from Tiny Images». En: págs. 32-33. URL: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Lewkowycz, Aitor et al. (2020). *The large learning rate phase of deep learning: the catapult mechanism*. arXiv: 2003.02218 [stat.ML].

Morshedifard, Ali et al. (2021). «Buckling of thermalized elastic sheets». En: *Journal of the Mechanics and Physics of Solids* 149, pág. 104296.

Myrtle.ai (2018). *How to Train Your ResNet 4: Architecture*. URL: https://myrtle.ai/learn/how-to-train-your-resnet-4-architecture/ (visitado 15-08-2021).

Ruiz-Garcia, M., Bonilla, L.L. y Prados, A. (2017). «Bifurcation analysis and phase diagram of a spin-string model with buckled states». En: *Physical Review E* 96.6, pág. 062147. DOI: 10.1103/PhysRevE.96.062147.

Ruiz-García, Miguel, Liu, Andrea J. y Katifori, Eleni (nov. de 2019). «Tuning and jamming reduced to their minima». En: *Physical Review E* 100.5. ISSN: 2470-0053. DOI: 10.1103/physreve.100.052608. URL: http://dx.doi.org/10.1103/PhysRevE.100.052608.

Ruiz-García, Miguel et al. (2021). *Tilting the playing field: Dynamical loss functions for machine learning*. arXiv: 2102.03793. URL: https://arxiv.org/abs/2102.03793.

Ruiz-Garcıa, Miguel, Bonilla, Luis L y Prados, Antonio (2016). «STM-driven transition from rippled to buckled graphene in a spin-membrane model». En: *Physical Review B* 94.20, pág. 205404. DOI: 10.1103/PhysRevB.94.205404.

Sagun, Levent, Bottou, Leon y LeCun, Yann (2017). *Eigenvalues of the Hessian in Deep Learning: Singularity and Beyond*. arXiv: 1611.07476 [cs.LG].

Sagun, Levent et al. (2018). *Empirical Analysis of the Hessian of Over-Parametrized Neural Networks*. arXiv: 1706.04454 [cs.LG].

Schwartz, Roy et al. (2019). *Green AI*. arXiv: 1907.10597 [cs.CY].

Shankar, Vaishaal et al. (2020). *Neural Kernels Without Tangents*. arXiv: 2003.02237 [cs.LG].

Soudry, Daniel y Carmon, Yair (2016). *No bad local minima: Data independent training error guarantees for multilayer neural networks*. arXiv: 1605.08361 [stat.ML].

Strubell, Emma, Ganesh, Ananya y McCallum, Andrew (2019). *Energy and Policy Considerations for Deep Learning in NLP*. arXiv: 1906.02243 [cs.CL].

Tandel, Aakash (agosto de 2017). *Support Vector Machines — A Brief Overview*. Ed. por Towards Data Science. [Online; posted 2-agosto-2017]. URL: https://towardsdatascience.com/support-vector-machines-a-brief-overview-37e018ae310f.

Toews, Rob (jun. de 2020). *Deep Learning's Carbon Emissions Problem*. Ed. por Forbes.com. [Online; posted 17-junio-2020]. URL: https://www.forbes.com/sites/robtoews/2020/06/17/deep-learnings-climate-change-problem/?sh=2c3e82806b43.

Xiao, Lechao et al. (oct. de 2018). «Dynamical Isometry and a Mean Field Theory of CNNs: How to Train 10,000-Layer Vanilla Convolutional Neural Networks». En: *Proceedings of the 35th International Conference on Machine Learning*. Ed. por Jennifer Dy y Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, págs. 5393-5402. URL: https://proceedings.mlr.press/v80/xiao18a.html.

Zhang, Chiyuan et al. (2017). *Understanding deep learning requires rethinking generalization*. arXiv: 1611.03530 [cs.LG].

Zoph, Barret y Le, Quoc V. (2017). *Neural Architecture Search with Reinforcement Learning*. arXiv: 1611.01578 [cs.LG].