Facultad de ciencias

Límites del aprendizaje Automático: Evaluación del rendimiento en condiciones restrictivas para la retención de información en métodos de Aprendizaje Continuo aplicado a Redes Neuronales Profundas

(Machine Learning limits: Performance evaluation in restrictive conditions for information retention regarding Continual Learning on Deep Neural Networks)

Trabajo de Fin de Grado para acceder al

Grado en Ingeniería Informática

Autor: Juan Castrillo Gutiérrez

Director: José Luis Montaña Arnaiz

Co-Director: Santos Bringas Tejero

Índice

Abstract

El aprendizaje continuo (CL) es una disciplina relativamente reciente que surge de la aplicación del aprendizaje automático mediante redes neuronales profundas, intentando solventar las limitaciones del aprendizaje para intentar adaptarse mejor a la forma de aprender que tienen los humanos.

En este trabajo se pretende, (1) mostrar algunas de las principales limitaciones de las redes neuronales profundas actuales, (2) describir el comportamiento en el aprendizaje de clases novedosas en el área de clasificación en visión por computador (2) definir una lista de objetivos de una solución de aprendizaje continuo CL ideal, (3) clasificar los principales enfoques para solucionar algunas de las limitaciones presentadas, (4) realizar una breve descripción del estado del arte en aprendizaje continuo, (5) presentar algunas de las formas de evaluar la efectividad de las propuestas del SoTA, (6) y finalmente discutir la viabilidad de la aplicación de las soluciones actuales.

The discipline of Continual Learning (CL) has recently been introduced to fix the limitations that have surge from the use of deep learning models, trying to breach the diference from the way humans learn.

In this work we intend to, (1) Show the constraints of trying to learn new classes applied to computer vision image classification, (2) define a list of desiderata for the ideal solution for the problems shown, (3) classify the main approaches to solve the CL problem, (4) briefly survey the state of the art, (5) explain some ways to evaluate a CL solution, (6) and finally discuss the state of CL current approaches

1. Introducción

El aprendizaje automático supervisado sufrió una explosión de popularidad con el desarrollo de las redes neuronales profundas. Muchos de los sistemas recientes no solo han alcanzado solventar tareas con un rendimiento casi humano sino que en algunas han conseguido alcanzarlo e incluso superarlo [1][2]. Estos resultados han permitido complementar, e incluso sustituir, personal humano en tareas de multitud de naturalezas. La velocidad a la que se expande su implementación está creciendo aún más.

Aún así, las redes neuronales profundas se encuentran siquiera en desarrollo y hay aun una gran cantidad de problemas que impiden avanzar 1.1.

Los modelos tradicionales, basados en redes de neuronas y propagación del error de una función de pérdida hacia atrás, tienen problemas para la destilación y retención de información. Son eficientes en la compresión de información de entradas a salidas dado un conjunto de datos, pero en ningún punto del proceso existen mecanismos capaces de abstraerse más allá del problema contenido en los datos de aprendizaje. Por ello, el rendimiento sobre datos no definidos en el problema (no relacionados) es pésimo y en muchos casos similar a una inicialización aleatoria.

El aprendizaje continuo es una evolución sobre el aprendizaje automático supervisado en redes neuronales profundas. Surge para solventar los problemas relacionados con el aprendizaje permanente, y permitir la retención de información en el tiempo, entre otros.

1.1. Límites de las Redes Neuronales

Las redes neuronales artificiales (ANN) aún presentan muchos problemas por resolver. Algunos de ellos han recibido extenso trabajo y tienen una serie de soluciones ya planteadas. A continuación se enumeran los principales problemas y/o limitaciones de las redes neuronales, junto con las soluciones más populares a cada una de ellas y en qué grado se han solventado:

(1) Dado que los detalles sobre la estructura en el aprendizaje de las redes neuronales aún no se conocen con exactitud, un problema enorme es cómo diseñar la arquitectura de la red. Por ejemplo, número de neuronas y de capas, interconexión entre capas, orden de las capas y la geometría de las neuronas dentro de cada capa. El problema consiste en encontrar la configuración óptima que conlleve el

menor tiempo de resolver sin sacrificar rendimiento. Debido a la gran cantidad de variables que supone intentar abordar el problema, aun no se ha encontrado una solución. Como regla general es necesario disponer de un número mayor de datos para obtener un resultado óptimo en redes grandes (como en Deep Learning), aunque un número inadecuado de capas o neuronas puede llevar a una situación de bajo aprendizaje (sobreajuste o subajuste). Para solventarlo se suelen utilizar redes propuestas (arquitecturas) y probadas del estado del arte [3], que son desarrolladas bajo prueba y error, adaptándolas a cada caso de uso con ligeras variaciones.

(2) A parte de la propia arquitectura, los diversos parámetros que afectan al proceso de aprendizaje (hiperparámetros) como son la tasa de aprendizaje, la función de pérdida, el optimizador, etc. deben ser ajustados acordemente para poder completar un entrenamiento óptimo y conseguir resultados deseables. Una selección no óptima generará diversos problemas, como puede ser un aprendizaje lento o un sobreajuste de los datos. Esta búsqueda resulta más sencilla (menos variables) que el anterior punto. Se puede resolver de la misma forma, siguiendo las recomendaciones dadas por la arquitectura seleccionada o mediante pruebas ensayo-error, además de intuición y experiencia previas. Además, existen métodos específicos para la optimización de hiperparámetros [4] como son, la búsqueda exhaustiva de hiper-parámetros mediante «grid search» o «random seach» que conllevan una altísima complejidad. También algunos más encaminados de menor complejidad por ejemplo métodos basados por gradiente o búsquedas bayesianas de probabilidad, cada uno con cierto grado de efectividad [5].

(3) Una limitación es en relación a la destilación de información de los datos de entrenamiento. El sobre-ajuste ocurre cuando una red se ajusta demasiado a los datos de entrenamiento, causando un pobre rendimiento sobre otros datos como puede ser los conjuntos de validación o test. Al contrario también existe el fenómeno de sub-ajuste, que ocurre cuando no se consigue aprender bien y resulta en mal rendimiento también sobre los propios datos de entrenamiento. Se puede solucionar consiguiendo aún más datos para que el conjunto de train sea más representativo de la imagen completa real. Esto se puede hacer recolectando más ejemplos (un ejemplo es una pareja de variables dependientes de entrada y su etiqueta (x_k, y_k)) o generándolos artificialmente mediante «data augmentation» [6] o utilizando otras redes neuronales como las GAN [7]. Otra solución es utilizar métodos de regularización como «dropout» o añadir términos de regularización L1-L2.

(4) El limitante principal en la aplicación de soluciones basadas en NN es la creación de conjuntos de datos. Generar un dataset conlleva conseguir los datos suficientes para que representen a la variedad de realidades que se quiere cubrir, que supone un esfuerzo muy grande de tiempo y personal (en muchos casos expertos), además de su etiquetado, procesado, limpieza y validación. Todo ello en el momento inicial antes de comenzar el proceso de creación de la solución.

(5) Finalmente el problema que ataña más al trabajo en cuestión que es el olvido catastrófico (ver 2.3). Resulta que un modelo ya entrenado no puede ser re-entrenado, por ejemplo para ampliarle o mejorar su rendimiento, sin disponer de todos los datos utilizados en el anterior entrenamiento y requiere de volver a entrenarlo con todos los datos además de los novedosos. Si se intenta entrenar solo con datos nuevos se perderá toda la información contenida en los datos que faltan.

Este (5) junto con (4) son los principales problemas que trata de abordar el aprendizaje continuo.

1.2. Motivaciones

Recientemente las redes neuronales han crecido en importancia, siendo las bases de los principales sistemas de decisión, búsquedas y destilación de información entre otros. Por su probada efectividad a resolver problemas han ido entrañándose más y más en el tejido industrial y social haciendo indispensable su uso. Este creciente interés en la automatización de procesos ha hecho surgir nuevas necesidades que van más allá de lo que puede ser resuelto mediante redes neuronales tradicionales.

Hay 2 campos para los que el aprendizaje continuo resulta muy interesante. Desde el punto de vista de la AI supone un nuevo paso hacia la creación de agentes inteligentes que puedan aprender continuamente. Por otra parte, más pragmático, este paradigma permite mejorar y simplificar la creación de los sistemas actuales. Faculta la adaptabilidad y escalabilidad.

Para este segundo, algunos de los requerimientos son:

- Ampliar el modelo con nuevos datos sin disponer de los anteriores o del tiempo, capacidad de computo o almacenamiento requerido para re-entrenar de nuevo.
- No se dispone de suficiente tiempo o esfuerzo para hacer una recolección de datos lo suficientemente exhaustiva como para conseguir el rendimiento deseado en el momento de despliegue.
- Los datos de los que se disponen tienen naturaleza temporal, se actualizan todo el tiempo y no pueden guardarse (streams).
- No se pueden definir los objetivos de aprendizaje en el momento del despliegue, ya sea porque el entorno es dinámico o muy complejo para conocerse.

1.3. Objetivos del aprendizaje continuo

El aprendizaje continuo lleva existiendo como concepto desde la concepción de las primeras redes neuronales. Supone la solución de los problemas del aprendizaje que plagan al estado del arte actual y que impide iterar sobre soluciones ya conseguidas, requiriendo que los investigadores creen arquitecturas novedosas para abordar nuevos problemas.

Aún siendo tan añejo el término, los objetivos de lo que supone un sistema CL ideal no han llegado a ser universalmente definidos. Con el creciente interés que ha surgido en esta última década se están haciendo esfuerzos por concretarlos como por ejemplo en [8].

A partir de [8] y una destilación de los enfoques de las soluciones propuestas en la literatura describo a continuación mi interpretación de que sería capaz de abordar un sistema de aprendizaje continuo ideal.

1. Permitir aprendizaje de futuras tareas no vistas anteriormente.
2. Retención de información con un acceso limitado (o sin acceso) a tareas anteriores. Mantener aprendizaje de tareas no disponibles.
3. Capacidad de crecimiento del modelo controlada (proporcional al número de tareas).
4. Destilar conocimiento entre tareas permitiendo mejorar el rendimiento en el tiempo.
5. Realizar un aprendizaje sin delimitaciones en las tareas por ejemplo uso de etiquetas de tarea. Debe ser capaz de trabajar con un stream de datos continuos sin conocer a que grupo pertenecen ni la estructura de los datos.

Como presentaré en este trabajo, esta definición del problema supone excesivamente difícil al ser tan distante del aprendizaje en redes neuronales actual, por lo que las primeras aproximaciones simplifican el problema para hacerlo abarcable. Principalmente tratan soluciones centradas exclusivamente en combatir el olvido catastrófico, pero el Aprendizaje continuo es mucho más amplio que eso.

2. Base Teórica

El concepto de aprendizaje continuo CL es formalmente propuesto en 1995 [9] y durante décadas ha sido solo una recopilación de ideas, desideratas y objetivos. Un ideal que pendía de la viabilidad de la evolución de las redes neuronales.

El incremento exponencial que han sufrido las metodologías de Machine Learning en términos de rendimiento y alcance ha abierto el camino para comenzar a trabajar hacia su desarrollo.

2.1. Definición del Problema

El aprendizaje continuo o permanente es un proceso en el que un sistema inteligente es dado una secuencia de tareas $t_0, t_1, ...$ siendo t_i la tarea número i que contiene k ejemplos $(x_{i0}, y_{i0}), (x_{i1}, y_{i1}), ...(x_{ik}, y_{ik})$ en forma fragmentada (varios ejemplos por tarea) o bien de manera continua $t_i = (x_i, y_i)$ que debe aprender en tiempo real o progresivamente.

En el punto $n+1$ del desarrollo, es decir, cuando se ha entrenado sobre $t_0, t_1, ..., t_n$ y se reciba una tarea t_{n+1}, el sistema debe (1) inferir lo aprendido sobre las n tareas anteriores para asistir con el aprendizaje, (2) aprender la instancia $n+1$ y al finalizar con ella, (3) actualizará su base

de conocimiento con la información destilada sobre t_{n+1}.

Consiste en 3 partes según [10],

1. Aprendizaje.
2. Destilación y almacenamiento de información del aprendizaje.
3. Utilizar información anterior para futuro uso.

2.2. Aprendizaje en redes neuronales

El aprendizaje automático mediante redes neuronales, en concreto aplicado a visión por computador, ha sido instrumental para propulsar la investigación de la inteligencia artificial estos últimos años. El paradigma actual consiste en ejecutar un algoritmo (red neuronal) sobre un dataset definido $D = \{(x_1, y_1), (x_2, y_2), ...\}$ hasta generar un modelo f_θ con el que poder predecir de manera satisfactoria salidas $f_\theta(x)$ a partir un conjunto de entradas x. Esta forma de aprender se denomina «Isolated Learning» [11].

ERM (Empirical Risk Minimization) [12] es un problema de optimización propuesto en 1998 que busca minimizar el error y encontrar los θ ideales para un f_θ óptimo. Es una simplificación del aprendizaje humano. Siendo D el dataset y D_{tr} el conjunto de entrenamiento, se busca minimizar

$$\frac{1}{|D_{tr}|} \sum_{(xi,yi)\in D_{tr}} \ell(f_\theta(x_i), y_i) \tag{1}$$

donde $\ell : Y \times Y \to [0, \infty)$ es la función de pérdida y $f_\theta(x_i)$ la predicción de la red para la entrada x_i. Requiere de varias pasadas por el conjunto de datos D_{tr} para llegar a optimizar.

Más concretamente aplicado a aprendizaje en redes neuronales profundas, una neurona es una función que recibe n entradas $x = \{x_1, x_2, ..., x_n\}$ las combina mediante el uso de pesos y un bias, generalmente $z = w \times x + b$ siendo $w = \{w_1, w_2, ...\}$ el conjunto de pesos para cada conexión de la neurona, y produce una salida $a(z)$ mediante un proceso denominado activación.

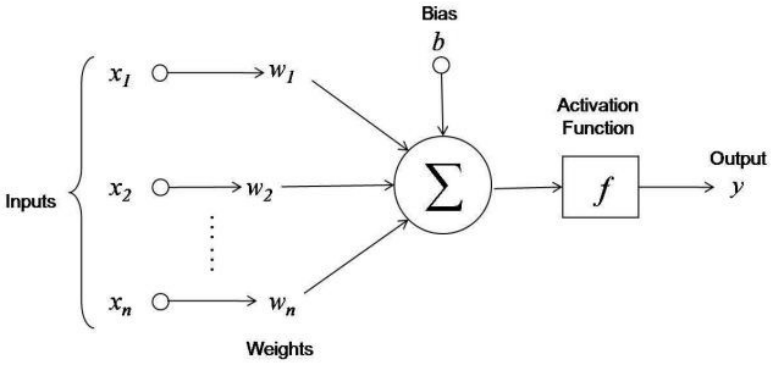

Figura 1: Diagrama de neurona simple donde w_i es el peso i para la entrada x_i y f es la función de activación que denominaremos a.

La activación es una función matemática que hace uso de una serie de variables que modifican la entrada, denominados pesos θ y bias b, para producir una salida en un rango de valores determinados. Por ejemplo:

La activación sigmoide contenida entre $[0, 1]$,

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

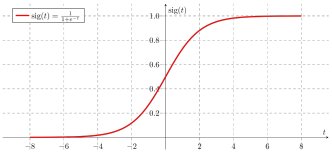

Figura 2: Función de activación sigmoide.

La activación ReLU contenida entre $[0, \infty)$ muy utilizada en arquitecturas modernas por su efectividad contra el efecto del «vanishing gradient» [13],

$$r(z) = max(0, z)$$

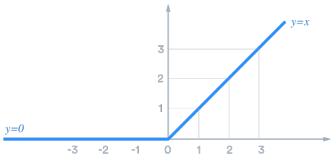

Figura 3: Función de activación ReLU.

La activación tanh contenida entre $[-1, 1]$ es una mejora en casi cualquier aplicación sobre la función sigmoide,

$$tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

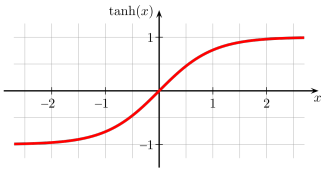

Figura 4: Función de activación tangente hiperbólica.

Una red neuronal consiste en una matriz multidimensional de neuronas ordenada en capas conectadas entre sí. Una capa es un grupo de neuronas normalmente agrupadas en forma de columna teniendo como entradas las salidas de la capa anterior y con sus salidas hacia la capa siguiente, pero puede variar como en el caso de las redes convolucionales (CNN) o las redes recurrentes (RNN). La capa inicial definida como la entrada es el punto de inserción de datos a la red, y la capa final es la que produce la salida final de la red, el resultado. Todas las capas intermedias son invisibles a la operación y se denominan «ocultas».

El objetivo del entrenamiento en una red neuronal es modificar los pesos de todas las neuronas tal que se minimice el error, calculado por la función de pérdida, haciendo que las salidas de la red coincidan con las esperadas lo más cerca posible. Este proceso es lo que se denomina aprendizaje y el método más óptimo actualmente para realizarlo es mediante la propagación del error , definido por la función de pérdida, hacia atrás, para conocer los gradientes de los pesos comúnmente conocido como «Backpropagation» junto con un optimizador para actualizar los pesos. Este proceso consiste en 4 pasos

1. Inferencia, uso de los pesos para modificar las entradas y producir salidas (2).
2. Evaluación del error, diferencia entre las salidas de la red y las esperadas mediante la función de pérdida ℓ.
3. Propagar el error desde la última capa (3) hasta la primera (hacia atrás) (4).
4. Actualizar los pesos de la red (6) para minimizar el error propagado mediante un optimizador (gradient descent) (1).

De manera genérica se puede definir las salidas de una capa en función de la anterior con sus pesos y bias como argumentos,

$$f_\theta(x) = [a^L(\theta^L, a^{L-1}(\theta^{L-1}, a^{L-2}(...), b^{L-1}), b^L)] \tag{2}$$

Donde $a^l(\theta^l, a^{l-1}(...), b^l)$ es la función de activación de la capa l con pesos θ^l, la activación de la capa anterior a^{l-1} y bias b^l. Donde θ^l es la matriz de pesos con los pesos relevantes para cada activación de la anterior capa $l-1$ sobre cada neurona de la capa actual l. Similarmente a^l es la matriz de activaciones de la capa l y b^l son los bias de todas las neuronas de la capa actual. L se refiere a la última capa que comprende la red (salida).

Siendo θ la matriz de todos los pesos de la red, $f_\theta(x)$ la predicción (o salida) de la red sobre unas entradas x, e y las predicciones correctas (o etiqueta) para esas mismas entradas x. El error viene derivado de la función de pérdida ℓ. En la última capa se define como,

$$\delta^L = \frac{\partial \ell}{\partial a^L} \frac{\partial a^L}{\partial z^L} \tag{3}$$

La función de pérdida o coste ℓ busca la diferencia entre los resultados esperados y y los producidos en la inferencia $f_\theta(x)$ dados unas entradas x,

El error sobre una variable z_i^l se calcula a partir de las derivadas parciales desde el error en la salida hasta la variable en cuestión,

$$\delta^l = \frac{\partial \ell}{\partial z^l} = \frac{\partial \ell}{\partial a^L} \frac{\partial a^L}{\partial z^L} \frac{\partial z^L}{\partial a^{L-1}} \cdots \frac{\partial a^l}{\partial z^l} \tag{4}$$

Más concretamente, a partir de los errores de la siguiente capa se pueden computar secuencialmente los errores sobre todas las capas, (el caso de una Feed-Forward Neural Network es muy similar al de una CNN),

$$\delta^l = \frac{\partial \ell}{\partial z^l} = \theta^{l+1} \delta^{l+1} \frac{\partial a^l}{\partial z^l} \tag{5}$$

Como se puede ver, la función de pérdida junto con las activaciones elegidas definen el gradiente a tomar para la modificación de los pesos. Dado que las funciones de pérdida tradicionales solo suelen tomar la salida de la red y la salida esperada como parámetros, solo es esa diferencia entre las predicciones y etiquetas de los datos disponibles la única información para poder evolucionar la red. Los algoritmos que utilizan esta información se llaman optimizadores. El más utilizado es el llamado «Stocastic Gradient Descent» (SGD). Realiza la corrección de los pesos mediante el descenso del gradiente producido por la función de pérdida sobre la diferencia entre las salidas esperadas y producidas, intentando minimizarla para cada paso del entrenamiento. La actualización de los pesos θ por tanto es,

$$\theta^l = \theta^l - \frac{\alpha}{|x|} (\delta^l a^{l-1}) \tag{6}$$

Similarmente con los bias b,

$$b^l = b^l - \alpha(\delta^l) \tag{7}$$

Siendo $\alpha \in [0, 1]$ el ratio de aprendizaje para suavizar el gradiente.

Otros optimizadores incluyen Adam [14] o SAM [15] entre muchos. Todos siguen el mismo principio añadiendo diferentes técnicas de optimización para limitar por ejemplo atascarse en mínimos locales o converger más rápido hacia una solución.

2.3. Olvido catastrófico

Las personas aprenden progresivamente, tarea a tarea. Esto supone que de un ejemplo de una tarea se destila información que se almacenará para utilizarla en el futuro y de cada ejemplo se aprenderá algo nuevo. Mientras que aprender múltiples tareas al mismo tiempo no resulta eficiente y para el caso de tareas complejas no es siquiera posible [16].

Por el contrario, las redes neuronales solo pueden aprender múltiples tareas en paralelo, de una vez. Intentar alimentar a la red como si fuera un humano, ejemplo a ejemplo (incrementalmente), resultaría en el aprendizaje y olvido de los ejemplos dados sin llegar a entender el proceso. La red nunca conseguiría un rendimiento bueno más allá de en el ejemplo dado en el momento, es decir, olvidaría lo que no se encuentre en el entrenamiento actual.

Dada la altísima especificidad con la que se ajustan al problema de las redes neuronales se podría pensar que para generalizar el modelo puede ser una buena idea entrenarlo de nuevo únicamente con los nuevos datos, como se realiza en [17]. Esta forma de transferencia de información es la utilizada en la disciplina de Transfer Learning [17]. Consiste en partir de un modelo entrenado sobre un conjunto de datos muy amplio para volver a entrenar sobre otro conjunto más específico facilitando el proceso e incluso mejorando los resultados [18].

El problema surge cuando solo se utilizan nuevos datos no vistos anteriormente por la red para el re-entrenamiento pero se desea mantener el rendimiento sobre todo el conjunto. Como se describe en el apartado 2.2, el aprendizaje en una red neuronal solo se realiza sobre los datos de los que dispone. La función de pérdida solo puede evaluar el rendimiento sobre los datos dados en el momento del entrenamiento. Por tanto, el gradiente se calculará para optimizar el rendimiento sobre estos, el conjunto de entrenamiento. Los datos no incluidos en el conjunto no se incluyen en la ecuación de pérdida a la hora de evolucionar la red en el proceso de aprendizaje. Por ello el rendimiento sobre datos pasados o futuros no se puede garantizar. Es equivalente a empezar con pesos aleatorios solo que quizá el entrenamiento sea algo más eficiente en algunos casos, véase Transfer learning.

Este problema se denomina Olvido Catastrófico, supone olvidar todo lo aprendido al optimizar para un nuevo problema.

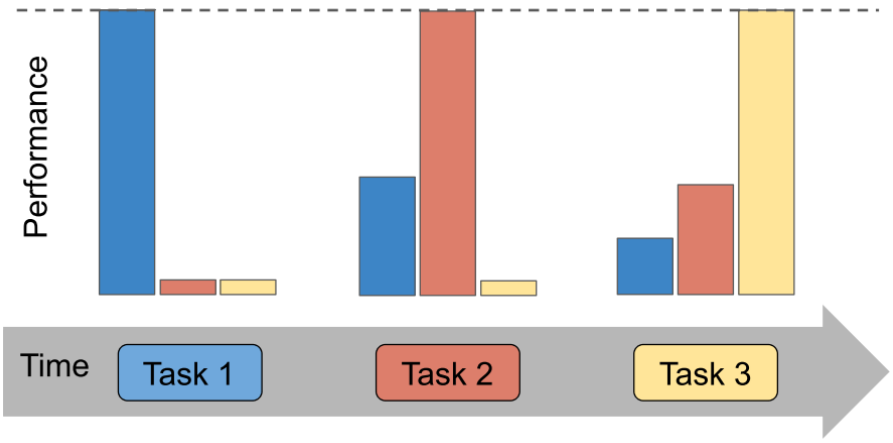

Figura 5: Ejemplo de Olvido Catastrófico sobre 3 tareas entrenadas una después de la otra. [19]

Las primeras definiciones del problema datan de finales del siglo XX [20][21], pero no será hasta recientemente que se concrete el problema y se encuentren maneras de mitigarlo.

Destacar la agrupación en columnas y la simplificación de notación siendo el superíndice el indicador de la capa a la que pertenece la variable y el subíndice (si se indica) la neurona a la que pertenece (dentro de la capa). Se han omitido los detalles sobre la multiplicaciones de las matrices y la agrupación además de las particularidades que conlleva la derivación de las ecuaciones ya que no es el propósito de este trabajo.

2.3.1. Stability-Plasticity Dilemma

El aprendizaje continuo rompe con la visión de el aprendizaje supervisado en cuanto a que no solo se busca optimizar un problema planteado si no también optimizar para los problemas que puedan llegar y los ya tratados. Todo ello sin sacrificar el rendimiento para la tarea actual.

El dilema de la estabilidad-plasticidad trata sobre el problema que surge al tratar de mejorar ambas. Optimizar para estabilidad reduce la plasticidad y viceversa, haciendo que sea imposible conseguir un modelo máximamente plástico y estable al mismo tiempo.

La **Estabilidad** de una red es la capacidad de mantener el rendimiento sobre lo ya aprendido (ejemplos entrenados) aún siendo mostrado ejemplos diferentes. Mientras que la **Plasticidad** se define como la capacidad de adaptarse a nuevos conocimientos.

Ambas propiedades se oponen entre sí. Cuanta más plasticidad, más fácil resulta aprender nuevas tareas pero más rápido olvida las ya aprendidas. Por otra parte, una red estable tiene mucha dificultad a adaptarse a nuevo conocimiento pero le será sencillo retener lo aprendido. La solución ideal consiste en encontrar un equilibrio entre ambas tal que se evite el olvido catastrófico y al mismo tiempo puedan expandirse las capacidades de la red plenamente.

3. Estado del Arte

La reciente explosión en interés en el aprendizaje continuo CL, especialmente aplicado aprendizaje supervisado en redes neuronales, ha llevado a un enorme progreso y novedosas ramas de investigación. Lamentablemente, por la velocidad vertiginosa de desarrollo científico, las bases no han sido establecidas completamente, a falta de terminología y objetivos comunes.

A pesar de todo el avance realizado en los últimos años, la disciplina de CL aún se encuentra en la infancia. Existen una enorme cantidad de diferentes ideas en la literatura, sin embargo, ninguna de las soluciones actuales resuelve de manera exitosa el problema del olvido y las soluciones que aparentemente lo consiguen realmente están esquivando el problema. «... a diverse set of approaches have been proposed in the literature ... However, these approaches impose different sets of simplifying constraints to the CL problem and propose tailored algorithms for the same. Sometimes these constraints are so rigid that they even break the notion of learning continually» - GDumb[22].

Partiendo de una revisión sistemática realizada por el estudiante de doctorado (y codirector de este trabajo) Santos Bringas Tejero del estado del arte desde el 2009 hasta finales 2021 (fecha de fin de búsqueda). Con un total de 103 artículos inicialmente, que fueron reducidos hasta unos 66 (criterios en el anexo 5.2). Comencé mi trabajo de filtrado que se dividió en 3 partes (1) Se clasificaron las propuestas en los métodos a los que pertenecen (ver 3.2), Surveys, reviews y estudios sobre propuestas concretas. (2) Se eliminaron los artículos que no trababan estrictamente soluciones de CL, es decir, todos los artículos que implementaban o utilizaban propuestas planteadas o planteaban ideas poco concretas. (3) Se añadieron artículos a partir de referencias de los presentes en el cribado inicial. Se eliminaron también soluciones que eran muy similares entre si o no planteaban un avance o conclusiones interesantes.

El resultado de ese trabajo concluyó en 5 reviews y/o Surveys y un listado de unas 60 propuestas relevantes clasificadas por los métodos que utilizan.

Date	Name	Paper Title	Arquitectural	Regularización	Revisión	Objetivos (no clasificación)
2014-11		Error-Driven Incremental Learning in Deep Convolutional Neural Network for Large-Scale Image Classification	✓			
2015-03	Distillation Networks	Distilling the Knowledge in a Neural Network		✓		
2016-05	RLSC	Incremental Robot Learning of New Objects with Fixed Update Time		✓		
2016-06	PNN	Progressive Neural Networks	✓			RL
2016-06	LWF	Learning Without Forgetting		✓		
2016-11	iCaRL	Incremental Classifier and Representation Learning	✓		✓	
2016-12	EWC	Overcoming catastrophic forgetting in neural networks		✓		
2017-01	PathNet	Evolution Channels Gradient Descent in Super Neural Networks	✓			
2017-04	SI	Continual Learning Through Synaptic Intelligence		✓		
2017-05	CWR	Core50 : a new dataset and benchmark for continuous object recognition	✓			
2017-05	DGR	Continual Learning with Deep Generative Replay			✓	
2017-06	GEM	Gradient Episodic Memory for Continual Learning	✓		✓	
2017-08	DEN	Lifelong learning with dynamically expandable networks	✓			
2017-08		Incremental Learning of Object Detectors without Catastrophic Forgetting		✓		
2017-11	MAS	Memory Aware Synapses: Learning what (not) to forget	✓			
2017-11	PackNet	Adding Multiple Tasks to a Single Network by Iterative Pruning	✓			
2017-12		Incremental Learning in Deep Convolutional Neural Networks Using Partial Network Sharing	✓			
2017-12	AG-GWR	Lifelong learning of human actions with deep neural network self-organization	✓			
2018-01	Piggyback	Adapting a Single Network to Multiple Tasks by Learning to Mask Weights	✓			
2018-02	Tree-CNN	A Hierarchical Deep Convolutional Neural Network for Incremental Learning	✓			
2018-02	SeNA-CNN	Overcoming Catastrophic Forgetting in Convolutional Neural Networks by Selective Network Augmentation	✓	-		
2018-05	LWF+	Keep and Learn: Continual Learning byConstraining the Latent Space for Knowledge Preservation in Neural Networks		✓		
2018-06	RWalk	Riemannian Walk for Incremental Learning: Understanding Forgetting and Intransigence		✓		
2018-06	AR1	Continuous Learning in Single-Incremental-Task Scenarios	✓	✓		
2018-09		End-to-End Incremental Learning		✓	✓	
2018-10	ADL	Autonomous Deep Learning: Continual Learning Approach for Dynamic Environments	✓			
2018-10	Conditional Replay	Marginal Replay vs Conditional Replay for Continual Learning			✓	
2018-11	GDM	Lifelong Learning of Spatiotemporal Representations With Dual-Memory Recurrent Self-Organization	✓		✓	
2018-12	GMED	Task-Free Continual Learning		✓		
2019-01	VCAE	Continual Representation Learning for Images with Variational Continual Auto-Encoder		✓		Autoencoder
2019-01	A-GEM	EFFICIENT LIFELONG LEARNING WITH A-GEM		✓		
2019-03	Global Distillation	Overcoming Catastrophic Forgetting with Unlabeled Data in the Wild		✓		
2019-04	ACE	Adapting to Changing Environments for Semantic Segmentation			✓	Segmentation
2019-04	DMC	Class-incremental Learning via Deep Model Consolidation		✓		
2019-04	DGM	Learning to Remember: A Synaptic Plasticity Driven Framework for Continual Learning	✓	✓		
2019-05	BiC	Large Scale Incremental Learning		✓	✓	
2019-07		Incremental Learning Techniques for Semantic Segmentation	✓			Semantic Segmentation
2019-07	Lifelong GAN	Continual Learning for Conditional Image Generation			✓	Generative Models
2019-07	AR1*	Rehearsal-Free Continual Learning over Small Non-I.I.D. Batches	✓	✓		
2019-08	MIR	Online Continual Learning with Maximally Interfered Retrieval			✓	
2019-08	ALASSO	Continual Learning by Asymmetric Loss Approximation with Single-Side Overestimation		✓		
2019-10	SIGANN	Self-Improving Generative Artificial Neural Network for Pseudorehearsal Incremental Class Learning			✓	
2019-11	WA	Maintaining Discrimination and Fairness in Class Incremental Learning		✓		
2019-11	UNLEARN	Lifelong Anomaly Detection Through Unlearning		✓		Anomaly detection
2019-12	PST	Single-Net Continual Learning with Progressive Segmented Training	✓			
2020-01		Efficient Incremental Learning Using Dynamic Correction Vector		✓	✓	
2020-02	CBCL	Cognitively-Inspired Model for Incremental Learning Using a Few Examples		✓		
2020-02	ER	Efficient Continual Learning in Neural Networks with Embedding Regularization		✓		
2020-03	iTAML	An Incremental Task-Agnostic Meta-learning Approach		✓		
2020-03	ESRIL	Exemplar-Supported Representation for EffectiveClass-Incremental Learning		✓		
2020-03	CGATE	Conditional Channel Gated Networks for Task-Aware Continual Learning		✓		
2020-04		Reducing catastrophic forgetting with learning on synthetic data			✓	
2020-06	Ova-INN	Continual Learning with Invertible Neural Networks	✓			
2020-11	DGFR	Class-Incremental Learning With Deep Generative Feature Replay for DNA Methylation-Based Cancer Classification			✓	Non Image Clasificaction
2020-11	GDumb	A Simple Approach that Questions Our Progress in Continual Learning			✓	
2021-10	CoPE	Continual Prototype Evolution: Learning Online from Non-Stationary Data Streams		✓	✓	

Esa lista se fue ampliando y reduciendo tras llegar a la colección final de 58 artículos que cubren desde los inicios del aprendizaje continuo «moderno» [23] en 2014 hasta el fin de la revisión sistemática con la última solución datando de alrededor del 2021 [24].

3.1. Simplificaciones

Debido a la altísima incertidumbre y complejidad que supone un paradigma de aprendizaje continuo, la literatura actual plantea acotar y simplificar el problema para poder abordarlo. Algunas de las principales son,

1. Simplificación en el manejo de los datos, ver 3.3.
2. Memoria para almacenar batches.
3. Entrenamiento offline en lugar de online.

3.2. Métodos de Aprendizaje Continuo

Los métodos de aprendizaje continuo toman como base modelos de aprendizaje profundo basados en redes neuronales generalmente en visión por computador aplicado a clasificación y los extienden en un intento de resolver alguno de los objetivos planteados en 1.3.

Desde el primer concepto de aprendizaje continuo se han propuesto múltiples formas de intentar abordar si no todos, alguno de los objetivos definidos en 1.3. Estos intentos han revelado los problemas que supone este enfoque y han hecho surgir un crisol de maneras de afrontarlos.

Las soluciones actuales pueden ser clasificadas en tres grupos según mi interpretación del SoTA. Esta forma de agrupar no es exclusiva, es decir, existen métodos que no pueden ser clasificados a una de ellas dado que utilizan varias. No obstante, esta división es bastante extendida en la literatura [25][26] y resulta de gran ayuda para conocer las tendencias de las soluciones actuales.

3.2.1. Revisión

Este grupo de soluciones surge de la idea trivial de solucionar el olvido catastrófico por falta de datos antiguos guardando datos antiguos y re-entrenando. Claramente esto no cumple los objetivos del aprendizaje continuo y las soluciones que toman este enfoque son más intrincadas. De manera general y bastante simplificada supone guardar de alguna manera las tareas de las que ya no se disponen para volver a entrenar la red sobre ellos. Puede ser una solución trivial como la realizada en GDumb [22] donde se guarda cierto numero de ejemplos restringido únicamente por la memoria disponible. O tan complejo como la creación de otros modelos que generen ejemplos «medianos» representativos de lo aprendido al haberlos visto todos [27].

Una clasificación algo más específica de este rango de enfoques podría ser,

(a) **Rehearsal** Guardan ejemplos vistos en el entrenamiento que se utilizarán en el futuro para volver a entrenar con ellos y recordarlos. Tiene problemas en el uso de memoria, falta de representación de los datos (al solo poder guardar unos pocos) y el problema de la selección de los mismos.

(b) **Generative Replay** Estos métodos consiste en crear estructuras que permitan generar datos para recordar los ya no disponibles. Suelen ser en forma de otras redes neuronales que destilen la información media de todo el conjunto a recordar.

(c) **Episodic Memory** Se dispone de una memoria limitada en el tiempo que solamente recoge ejemplos del stream para poder entrenar y debe ser limpiada al terminar de con el bloque de datos recogido.

Suelen tener problemas de overfitting a los sistemas de recolección de datos anteriores. Además que suponen un mayor coste computacional al requerir volver a entrenar ejemplos ya vistos.

3.2.2. Regularización

Buscan regular la importancia en el aprendizaje de clases nuevas sobre las ya vistas. Suavizan la forma en la que se actualizan los pesos de la red, promoviendo la estabilidad y reduciendo

la plasticidad con la intención de limitar el olvido catastrófico. Esto se consigue modificando el proceso de optimización, extendiendo la función de pérdida para intentar consolidar conocimiento.

Algunos enfoques triviales son el uso de «dropout», eliminar conexiones entre capas , early stopping, parar el entrenamiento cuando el rendimiento no mejore, o «Weight Sparsification», eliminar pesos para reducir complejidad y acelerar el entrenamiento.

En general es el método más eficiente computacionalmente ya que no aumenta el número de operaciones más que un procesado en la fase de cálculo del error, pero sacrifican el aprendizaje a largo plazo especialmente con tareas similares ya que no hay mecanismo de recuerdo, solo se limita el olvido.

3.2.3. Arquitecturales

Parten de la idea que tras entrenar sobre una tarea los pesos aprendidos son los ideales para ella luego es requerido conservarlos. La forma de guardar ese conocimiento se puede llevar a cabo (1) congelando pesos antes de cada nuevo bloque y/o (2) modificando la arquitectura del modelo [28], asegurando que ciertas partes de la red correspondan con ciertas tareas. por ejemplo, expandiendo el modelo por cada bloque (más capas, aumentar la última capa de clasificación, ...). Estos últimos tienen la desventaja de aumentar el coste computacional rápidamente por el explosivo aumento en pesos a entrenar, con nuevos pesos cada tarea novedosa.

Es un punto intermedio entre usar una red estática y utilizar una red extra para cada nuevo problema encontrado.

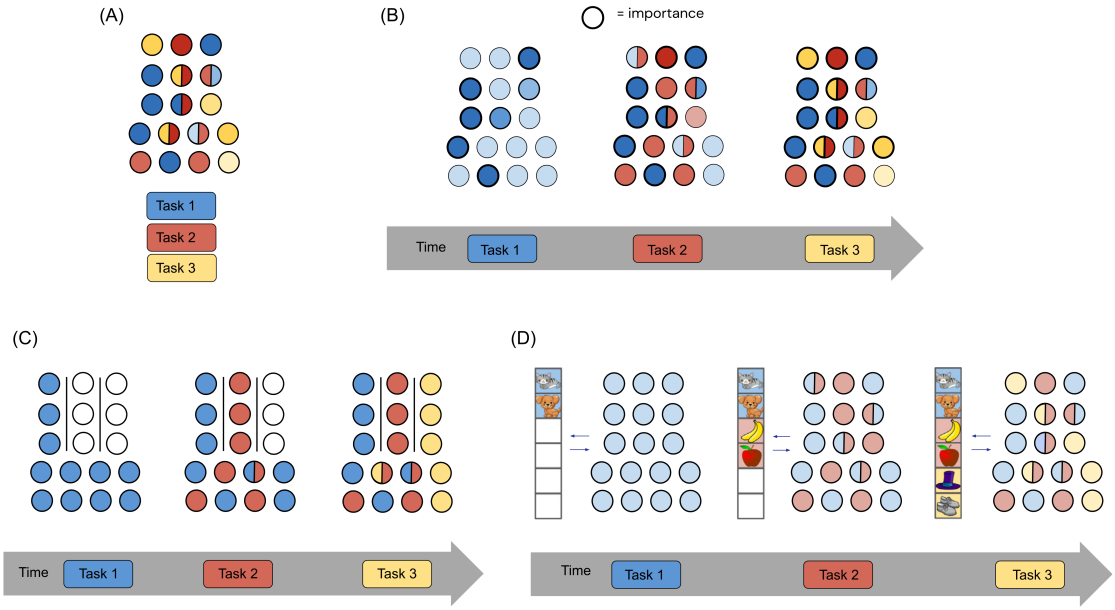

Figura 6: Métodos de aprendizaje Automático. (A) Método Tradicional, todas las tareas de una vez. (B) Regularización, los pesos se reutilizan para múltiples tareas. (C) Aislamiento de parámetros, cada tarea tiene unas neuronas y pesos exclusivas para ellas. (D) Replay, Se dispone de una memoria extra para almacenar información de entrenamientos anteriores y reentrenar. [19]

3.3. Manejo de datos

La creación de un conjunto de datos para aprendizaje supervisado en entrenamiento tradicional conlleva la recopilación de una gran cantidad de ejemplos que cubra la totalidad del problema a abordar. Esto supone un gran problema en la práctica ya que requiere de una alta cantidad de tiempo y esfuerzo de etiquetado por parte de expertos. Lo ideal sería poder aprender de ejemplos en el momento que se consigan (según se disponga de ejemplos) sin que supongan ningún degradado en el resultado final (con respecto al rendimiento si no estuvieran divididos).

La manera de alimentar datos planteada entre los objetivos de CL es un stream de datos continuo, es decir, una especie de tubería de datos por la que las entradas no se almacenan, llegan en tiempo real a los destinatarios y se descartan inmediatamente. Su función sería en cada momento traer toda la información del entorno al agente inteligente. Este paradigma de datos continuos sin estructura resulta demasiado complejo para ser abordado en su totalidad. Por ello las investigaciones actuales (estado del arte) se han decantado por una serie de simplificaciones. La más común consiste en dividir el entrenamiento en bloques de datos llamados tareas y separados en función de las etiquetas. Las tareas se encuentran disponibles sin límite de tiempo (offline). Se denomina Task-Incremental (TI) que supone una división clara donde cada ejemplo se etiqueta con el identificador del bloque al que pertenece y se puede realizar de 3 formas distintas,

(a) **Nuevas Instancias (NI)**

Las tareas contienen únicamente nuevos ejemplos de las clases ya conocidas. No se introducen ninguna nueva clase. Una clase es una etiqueta concreta y_k. En el contexto de clasificación en visión por computador son todas aquellas fotos que reciben la misma etiqueta y_k.

No están ideados para extender el modelo si no entrenar en este paradigma permite evolucionar sobre lo ya conocido mejorando el rendimiento en el tiempo.

(b) **Nuevas Clases** (NC)

Cada bloque contiene ejemplos de una o varias nuevas clases que no han sido previamente incluidas en ningún bloque anterior y nunca más serán enseñadas. Todos los ejemplos de las clases deben estar contenidas en el bloque en el que se introducen (no hay una segunda oportunidad).

(c) **Nuevas Instancias y Clases (NIC)**

No hay limitaciones sobre que pueden contener los bloques, más allá de no repetir ejemplos repetidos (mismo ejemplo en bloques diferentes).

Otra alternativa se denomina Class-Incremental (CI) que como Task-Incremental (TI) sufre de las mismas divisiones posibles pero omite la identificación del bloque en el ejemplo (x_i, y_i) en lugar de (x_i, y_i, t_i) con la intención de representar una situación realista ya que no se puede asumir una entrada ordenada y estructurada de datos. Algunos sistemas lo utilizan pero supone un importante aumento en complejidad.

Entre los tres formatos de tareas posibles, la más compleja es NIC ya que se tiene que balancear continuamente la plasticidad con la retención de información. Mientras que la más sencilla resulta ser NI ya que ya se conocen las clases y solo se tiene que controlar el olvido de los ejemplos anteriores. En la casi totalidad de las soluciones se trata con NC. Supone la más interesante en la práctica porque permite ampliar el dominio de modelos sin ser tan compleja como NIC.

Estas divisiones de datos en tareas se pueden alimentar al modelo de 2 maneras distintas en función de cómo se agrupen,

(a) **Single incremental task (SIT)**

Cada tarea se aísla y se entrena por separado sin contexto de las demás. Es el caso más complejo y que menor rendimiento consigue

(b) **Multi Task (MT)**

Se toman múltiples tareas que entrar al mismo tiempo para cada etapa del proceso.

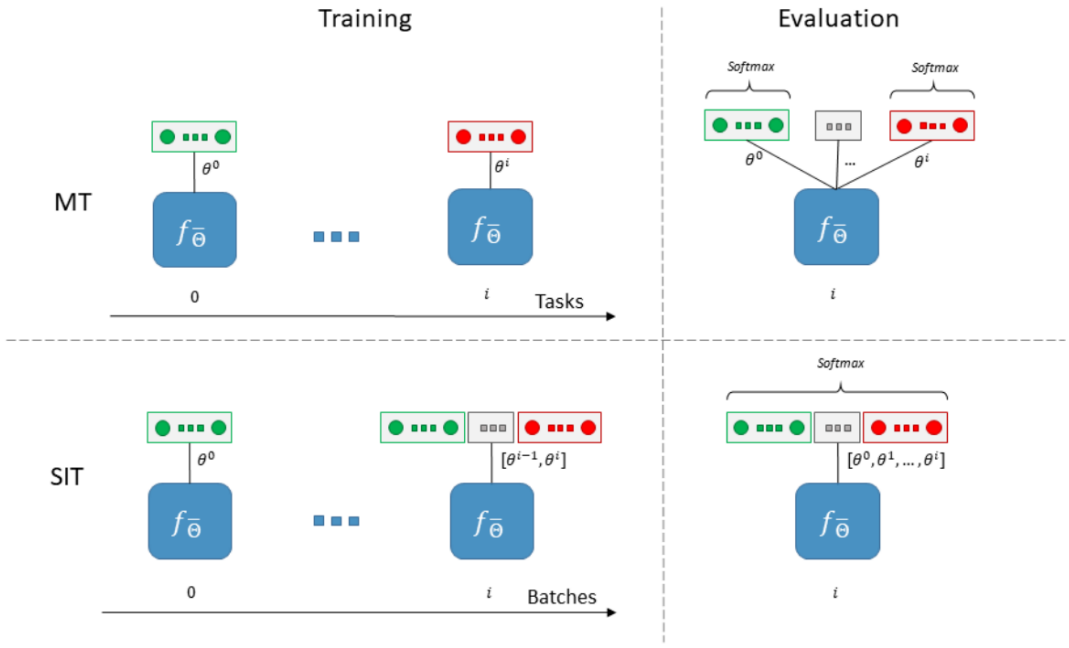

Figura 7: Proceso de entrenamiento y test para Single Incremental Tasks (SIT) y Multitask (MT)

3.4. Soluciones CL

Durante el proceso de revisión sistemática fue claro que había una estructura jerárquica de soluciones basadas en otras anteriores que me resultó muy interesante reflejar.

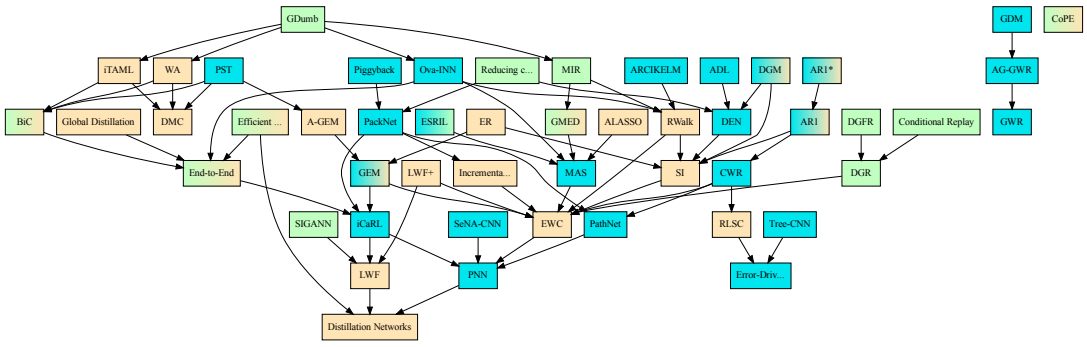

Figura 8: Árbol de inspiraciones de las soluciones planteadas coloreados por metodología. Naranja: Regularización, Turquesa: Arquitectural y Verde: Revisión.

La jerarquía resultante no es objetiva, es basada en mis criterios de similitud entre propuestas ayudado por las menciones de los autores a trabajos previos o inspiraciones. El árbol completo generado con las referencias cruzadas entre artículos es demasiado complejo y se encuentra adjuntado en el anexo 25. A partir de él se hizo una gran simplificación ignorando referencias en caso de disponer de otras más recientes basadas en estas o si las referencias eran solo como contexto y no tienen demasiada relación sobre lo planteado. Además se ha priorizado la lectura sobre el rigor de similitud.

El análisis y resumen de todas las propuestas es demasiado trabajo para ser desarrollado en esta memoria. Por tanto voy a mostrar una selección de tres métodos con rendimiento SoTA, que no han sido evaluados juntos anteriormente, y las inspiraciones que han tomado desde el inicio hasta la actualidad. He seleccionado AR1* [29] representando los métodos de regularización y arquitectura, CoPE [24] como revisión y GDumb [22] como crítica al estado del arte. Los resultados obtenidos por

estas 3 soluciones junto con una breve descripción se encuentran en el anexo 5.3. Además incluiré una breve descripción de alguno de los artículos restantes como un índice para su investigación futura.

Las soluciones suelen tener un sobrenombre que consiste en una o pocas palabras que normalmente forman las siglas del enfoque utilizado.

3.4.1. AR1*, Rehearsal-Free Continual Learning over Small Non-I.I.D. Batches

[29] **AR1*** presentado a mitad de 2019 se presenta como un método de aprendizaje continuo con un rendimiento excepcional «AR1* can outperform other state-of-the-art rehearsal-free techniques by more than 15 % accuracy in some cases» - [29] y muy ambicioso, «Ideally, continual learning should be triggered by the availability of short videos of single objects and performed on-line on on-board hardware with fine-grained updates» - [29]. Surge como evolución al anterior método AR1 [26] que ya conseguía extender el estado del arte con su rendimiento excepcional y su cobertura de casos de uso.

Toda esta rama tomó como base el algoritmo de Copy Weights with Reinit (**CWR**) [30] presentado junto a al benchmark CORe50.

Este método es una de las primeras instancias de modificaciones de arquitectura. Consiste en una síntesis sobre PNN [28] que resulta en un método más simple y ligero aunque algo menos flexible que evita el crecimiento descontrolado de parámetros. En comparación supone un coste muy bajo y permite el entrenamiento de mayores series de datos. Pretende controlar el olvido mediante la expansión de la última capa con tantas neuronas (de pesos aleatorios) como clases novedosas se encuentren. Y manteniendo los pesos del primer batch de entrenamiento para toda la red menos la capa de salida.

```
cw = 0
init T random or from pre-trained model
for training batch Bi:
    expand output layer with si neurons for the new clases in Bi
    for all neurons in output layer:
        random re-init tw
    Train model with SGD on si classes of Bi:
        if Bi is the first batch: learn T and   tw
        else:                     keep  T learn tw
    for each class j in si:
        cw[j] = wi * tw[j]
    Test the model using T and cw
```

Figura 9: Algoritmo CWR. Siendo T los pesos de todas las neuronas. tw la matriz de pesos de las neuronas de la última capa usados para entrenamiento, cw matriz de pesos de las neuronas de la última capa usados para inferencia y B_i un bloque de datos

Este se extiende en **CWR+** planteado por [26] que propone cambiar dos detalles,

1. Inicialización a ceros en lugar de aleatoria de la última capa en cada batch $tw = 0$.
2. Los pesos de la última capa pasan a ser desplazados sobre la media de los pesos en la última capa, «mean-shift», $cw[j] = tw[j] - mean(tw)$.

CWR+ demuestra ser bastante efectivo limitando el olvido en NC-SIT pero pierde la capacidad de adaptarse al congelar los pesos de todas las capas menos la última.

Teniendo en cuenta esta limitación se propone como alternativa **AR1** [26], aumenta su alcance permitiendo el entrenamiento de todos los pesos de la red mediante el uso de regularización con el propósito de tolerar el movimiento de pesos críticos pero de manera limitada. La regularización planteada para hacerlo utiliza las ideas de Synaptic Inteligence (SI) [31] para dirigir el aprendizaje de los pesos de la red en lugar de mantenerlos estáticos para todo el entrenamiento. Éste, al igual que EWC [32], utilizan una matriz de información de Fisher para evaluar la relevancia de los pesos de la red, relevante a la hora de actualizarlos.

```
cw = 0
init T random or from pre-trained model
To = 0
Fw = 0
for training batch Bi:
    expand output layer with si neurons for the new clases in Bi
    for all neurons in output layer: tw = 0
    Train model with SGD on si classes of Bi:
        learn tw (without regularization)
        learn T  (using SI regularization with To and Fw as params)

    for each class j in si:
        cw[j] = tw[j] - avg(tw)
    To = T
    Update Fw with Bi trajectories
    Test the model using T and cw
```

Figura 10: Algoritmo AR1. Donde To es la matriz de pesos óptimos compartida con el entrenamiento anterior, y Fw es la matriz de relevancia de pesos utilizada para la regularización.

La actualización de Fw se realiza mediante el método descrito en SI.

$$Fw = clip(F, max(F)) \tag{8}$$

donde F es la matriz de variaciones de los pesos durante el entrenamiento con componentes $F_k, \forall \theta_k \in T$ denominada matriz de información de Fisher, y θ_k es un peso cualquiera de la red T. Además TM_k supone el movimiento total del peso θ_k durante el entrenamiento de un bloque de datos.

$$F_k = \frac{\sum \Delta L_k}{TM_k^2 + C}$$

Mientras que ΔL_k se define como la variación de la pérdida definida en el entrenamiento para un peso cualquiera θ_k, $\Delta L_k = \Delta\theta_k \frac{\partial L}{\partial \theta_k}$. y C es una constante pequeña para evitar dividir por 0.

Por otra parte, la regularización SI consiste en modificar la manera en la que se actualizan los pesos para tener en cuenta los entrenamientos pasados. La actualización de los pesos por tanto se reduce a

$$\theta_k = \theta_k - \eta \frac{\partial L}{\partial \theta_k} - \eta F_k(\theta_k - \theta_k*)$$

donde L es la función de pérdida del método, θ_k^* es el peso óptimo discutido en el algoritmo 11, $\theta_k^* \in To$ y η el ratio de aprendizaje clásico.

Finalmente, se presenta CWR*, la base de AR1*, que funciona sobre formato NC y NIC. Se complica ligeramente respecto a CWR+ añadiendo factores para transferir información del entrenamiento entre grupos de datos además de ajustar las re-inicializaciones de pesos en la última capa.

```
cw = 0
past = 0
init T random or from pre-trained model
for training batch Bi:
    expand output layer with si neurons for the new clases in Bi
    for all neurons j in output layer:
        if class j in Bi: tw[j] = cw[j]
        else:             tw[j] = 0
    Train model with SGD on si classes of Bi:
        if Bi is the first batch: learn T and   tw
        else:                     keep  T learn tw
    for each class j in Bi:
        wpast[j] = sqrt(past[j] / count of j in Bi)
        cw[j] = (cw[j] * wpast[j] + (tw[j] - mean(tw))) / wpast[j]+1
        past[j] += count of j in Bi
    Test the model using T and cw
```

Figura 11: Algoritmo CWR*.

AR1*, al igual que AR1 sobre CWR+, extiende CWR* añadiendo el aprendizaje de parámetros de la red T con regularización SI y la actualización de parámetros necesarios para hacerlo.

Como anexo, PNN [28] es una de los primeras estrategias arquitecturales que utiliza la combinación de dos técnicas, congelación de parámetros y expansión de la red. Los autores mostraron que podía llegar a ser efectivo en pequeñas series de tareas simples pero tiene la gran limitación de que la forma usada para expandir la red resulta en un crecimiento del número de parámetros no controlable para series de mayor longitud.

3.4.2. CoPE, Continual Prototype Evolution, Learning Online from Non-Stationary Data Streams

CoPE [24] es una estrategia de aprendizaje continuo online basada en un sistema «learner-evaluator» con una metodología de revisión.

Tiene 3 componentes principales

1. Representaciones evolutivas
2. Revisión balanceada
3. Función de pérdida PPP (pseudo-protopical proxy loss)

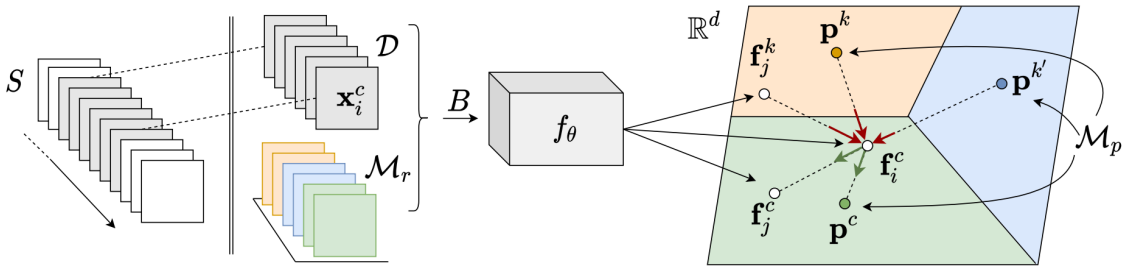

Figura 12: Arquitectura de CoPE.

(1) Se dispone de una memoria $\mathcal{M}$ para retención de ejemplos durante el entrenamiento. Se divide en memoria de revisión $\mathcal{M}_r$ que contiene varios ejemplos por clase y «prototypical memory» $\mathcal{M}_p$, una pequeña memoria usada para el cálculo de la pérdida.

El primer componente consiste en seleccionar el ejemplo ideal denominado prototipo p^c que guardar en $\mathcal{M}_p$ para representar a la clase c de entre todas las clases encontradas $c \in \mathcal{Y}$. El clasificador nearest neighbor encuentra el prototipo más similar dado una entrada x_i,

$$c' = max_{c \in \mathcal{Y}} f_i^T p^c$$

Siendo f_i^T el clasificador para una query x_i sobre todas las clases T. Estos prototipos tienen que evolucionar en el tiempo para evitar quedarse obsoletos con los datos del stream sin olvidar las representaciones anteriores. Para ello se actualizan cada batch usando una corrección de centro de masa,

$$p^c = \alpha \; p^c + (1 - \alpha) \frac{1}{|B^c|} \sum_{x^c \in B^c} f_\theta(x^c) \tag{9}$$

Siendo B^c el bloque de datos que contiene la clase c, $B^c = \{(x_i, y_i) \in B \mid y_i = c\}$ y x^c todas las entradas en B^c. Tras cada actualización del prototipo en 9 es necesario normalizar con L2.

(2) En cuanto al proceso de revisión, la memoria $\mathcal{M}_r$ es fija y se divide uniformemente sobre el número total de clases observadas $|\mathcal{Y}|$ de forma dinámica M_r^c.

(3) La función de pérdida se basa en las distancias intraclass e interclass, que son posibles de calcular al disponer de el esquema de clases como clusters en el espacio R^d con centros el prototipo p^c para cada $c \in \mathcal{Y}$. Se construye un conjunto de atractores («prototypical atractors») y repulsores («prototypical repellors») para cada instancia x_i^c usando el prototipo p^c de la clase y otras instancias en B. Los atractores de la clase c y conjunto B se forman como la unión del prototipo con el conjunto de pseudoprototipos de la clase c, $\hat{p}^c$ que son todos aquellos ejemplos en B que pertenecen a c y no son el ejemplo en cuestión x_i^c,

$$\mathbb{P}_i^c = \{p^c\} \cup \{\hat{p}_j^c = f_\theta(x_j^c) \mid \forall x_j^c \in B^c, i \neq j\}$$

El conjunto de repulsores se forma con la representación de x_i^c y el prototipo de su clase c,

$$U_i^c = \{p^c, \hat{p}_i^c = f_\theta(x_i^c)\}$$

A partir de los 2 conjuntos se puede formular un problema de clasificación binaria, con la unión de las probabilidades de que el ejemplo x_i^c se prediga como c y las instancias x_j^k no se predigan como c todos los ejemplos en B,

$$P_i = P(c|x_i^c) \prod_{x_j^k} (1 - P_i(c|x_j^k))$$

Con las probabilidades dadas por la esperanza de encontrar esos ejemplos en los conjuntos de atracción y repulsión,

$$\begin{cases} P(c|x_i^c) & = \mathbb{E}_{\tilde{p}^c \in \mathbb{P}_i^c}[P(c|f_i^c, \tilde{p}^c)] \\ P_i(c|x_j^k) & = \mathbb{E}_{\tilde{p}^c \in U_i^c}[P(c|f_j^k, \tilde{p}^c)] \end{cases}$$

Donde $\tilde{p}^c$ es un proxy para la media latente de la clase c y τ es una constante para controlar el nivel de concentración de la distribución,

$$P(c|f, \tilde{p}^c) = \frac{\exp\left(f^T \tilde{p}^c / \tau\right)}{\exp\left(f^T \tilde{p}^c / \tau\right) + \sum_{k \neq c} \exp\left(f^T p^k / \tau\right)}$$

Finalmente, la función de pérdida se puede definir como la esperanza logarítmica negativa sobre todas las instancias de B,

$$\mathcal{L} = -1/|B| \left[\sum_i \log P(c|x_i^c) + \sum_i \sum_{x_j^k} \log(1 - P_i(c|x_j^k)) \right] \tag{10}$$

En resumen, CoPE consiste en tomar ejemplos de un stream de datos de la forma descrita en (1). Re-entrenar el modelo con datos recogidos (2) y finalmente aplicar la función de pérdida para regularizar la actualización de pesos mediante (3).

```
Require:
    data-stream S,
    memory capacity Mc,
    learning rate η
Initialize
    replay memory M = [],
    prototype memory P = [],
    observed classes C = [],
    model parameters θ random or from pre-trained model

for Bn = [(x1, y1), ..., (xB, yB)] in S:
    BM = take size(Bn) random samples from M
    for (xi, yi) in [Bn union BM]:
        if yi not in C:
            initialize the class yi
        add fθ(xi) to the batch Bn predictions
    P = update Prototypes
    M = update Memory
    J = Add the PPPloss for all the predictions in Bn
    Update weights with J: θ = θ + η ∇J
```

Figura 13: Algoritmo CoPE.

3.4.3. GDumb, A Simple Approach that Questions Our Progress in Continual Learning

GDumb [22] «Greedy Sampler and Dumb Learner», surge como una crítica a las soluciones planteadas en el SoTA. Realizan un survey detallando las simplificaciones, problemas y enfoques del momento. De forma bastante crítica concluyen que la situación actual es preocupante y que apenas se ha realizado progreso real en el área de CL, «The fact that GDumb, even though not designed to handle the intricacies in the challenging CL problems, outperforms recently proposed algorithms in their own experimental set-ups, is alarming. It raises concerns relating to the popular and widely used assumptions, evaluation metrics, and also questions the efficacy of various recently proposed algorithms for continual learning.» - [22]. Para probar sus acusaciones plantean un método que ellos declaran que aún sin estar diseñado para solventar el problema de CL consigue superar a la mayoría (si no todas) las soluciones en el momento de su publicación, «We show that even though GDumb is not specifically designed for CL problems, it obtains state-of-the-art accuracies (often with large margins) in almost all the experiments when compared to a multitude of recently proposed algorithms.» - [22].

Este método planteado se basa en un recolector de ejemplos voraz y un aprendizaje sin modificaciones como se hace alusión en su nombre. Opcionalmente se hace uso de una máscara para inferir sobre CI o TI.

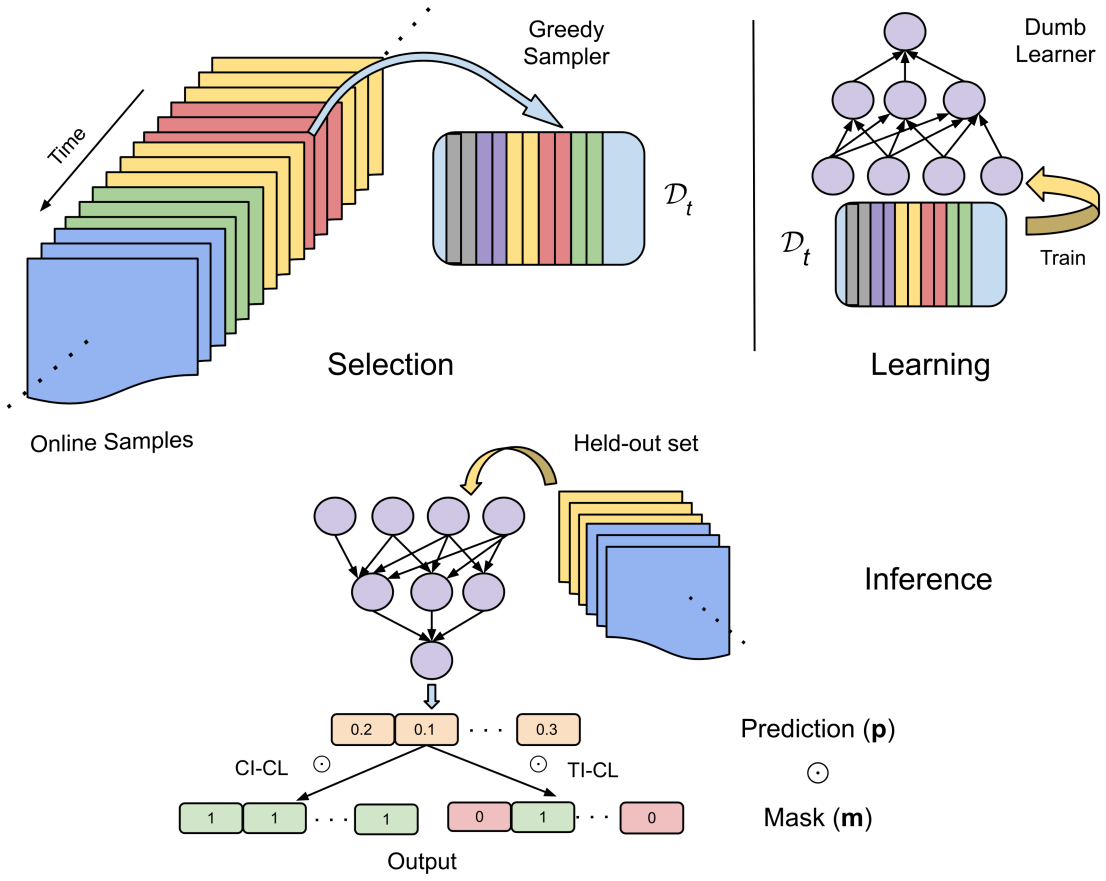

Figura 14: Arquitectura de GDumb.

Dado un límite de memoria, el recolector guarda ejemplos del stream de datos en un conjunto D_t hasta llenarla, con como única restricción el balanceo de la distribución de clases recogidas. Cuando encuentra una nueva clase, se crea un nuevo subconjunto para esa clase y se limpian ejemplos guardados de otras clases anteriores (aleatoriamente) para hacer hueco en memoria.

```
Require:
    k # Total memory capacity
    Y # Set of seen clases
    D # Memory of stored samples, D[yi]: Samples saved for class yi
    (xt, yt) # current example

kc = k / Y # memory capacity for class yt
if yt not seen before or count(D[yt]) < kc:
    if memory is full:
        yr = select class with most examples from D:
        (xi, yi) = select random sample from D[yr]
        remove (xi, yi) from D[yr]
    add (xt, yt) into D[yt]
    add the current class to Y

return D
```

Figura 15: Algoritmo del Greedy Sampler.

A partir de la selección, se realiza el entrenamiento únicamente con los ejemplos seleccionados en D_t de manera tradicional (en la implementación oficial se utiliza SGD).

3.4.4. Otros

Entre los inicios de modelos arquitecturales encontramos **GNG** (Growing Neural Gas) [33] (1995) un modelo pionero que se basa en añadir neuronas a intervalos fijos durante el entrenamiento para minimizar los errores locales. A partir de él surge **GWR** [34], propuesto en 2002, que sustituye el criterio de intervalos fijos por un estudio de actividad. Se añaden más neuronas a partir de un límite mínimo de activación global de las neuronas entrenadas de la red.

En [35] se presentan los primeros intentos ingenuos disponibles contra el olvido catastrófico antes de su aparición,

	Fine Tuning	Duplicating and Fine Tuning	Feature Extraction	Joint Training	Learning without Forgetting
new task performance	good	good	X medium	best	✓best
original task performance	X bad	good	good	good	✓good
training efficiency	fast	fast	fast	X slow	✓fast
testing efficiency	fast	X slow	fast	fast	✓fast
storage requirement	medium	X large	medium	X large	✓medium
requires previous task data	no	no	no	X yes	✓no

Figura 16: Trabajos contra el olvido catastrófico previos según LWF.

Brevemente los describo. **Feature extraction** no modifica los pesos pero recicla salidas de la red como entradas. **Fine-Tuning** mantiene fijos los parámetros específicos para tareas anteriores mientras que permite modificar los demás aunque con un ratio de aprendizaje bajo. **Joint Training** es el proceso de optimizar todos los parámetros de la red al mismo tiempo. Una implementación primitiva de revisión.

También se presenta una nueva solución con el propósito de mitigar los inconvenientes planteados por las anteriores, denominada **LwF**. Es la primera estrategia de regularización. Se enfoca principalmente en intentar solventar el olvido Catastrófico, tendencia que se mantendrá cierta en las evoluciones de esta metodología. Pretende preservar la precisión del modelo sobre tareas ya vistas imponiendo estabilidad mediante destilación de conocimiento. De forma muy simplificada, antes de entrenar cada tarea, se crean pseudo-etiquetas llamadas LwF-logits que se utilizan para entrenamiento junto con las etiquetas reales. Esto hace que se pueda ajustar la pérdida a la hora de realizar la propagación hacia atrás ajustando esas pseudo-etiquetas. De él, surge el método con diferencia más citado de la literatura de aprendizaje continuo, **EWC** [32], es una evolución de la técnica de regularización. Utiliza la matriz de información de Fisher para mediante máscaras limitar la evolución de ciertos pesos. Como ya comentado, **SI** [31] mejora en todos los sentidos a EWC utilizando un método muy similar.

Siguiendo esta linea, se propone **iCarl** [36] mejora LwF en varios aspectos. Añade una estrategia de revisión que guarda datos de entrenamiento dinámicamente hasta un tamaño dado, lo llaman «exemplar set» para preservar el rendimiento en clases anteriores. Al mismo tiempo se presenta [37] que no llega a conseguir mucha tracción por la complejidad de su implementación y bajo rendimiento. Utiliza la descomposición de Cholesky para la actualización de los pesos.
Esta estrategia de combinar revisión con regularización se ve aumentada en **GEM** [38]. Es una técnica que regulariza el modelo en función de los gradientes calculados en la propagación de ejemplos de tareas pasadas sobre la red actual. Estos gradientes son forzados a tener la misma dirección vectorial que en el momento de entrenar la tarea pasada. Además, guarda un conjunto de tareas pasadas en una memoria de tamaño fijo con la que evita olvidar aun más. Ambos métodos se utilizan en conjunto para proyectar los gradientes hacia un mínimo local de rendimiento sobre todas las tareas. Una nueva versión, **A-GEM** [39] asegura que a cada paso no se pierda rendimiento medio de la memoria episódica de GEM. Con ello se gana mejor precisión media pero se sacrifica el olvido en el peor caso ya que este permite pérdida en ciertas tareas a cambio del rendimiento medio al contrario que GEM que fuerza una mejora en todas. Partiendo de LwF se plantea LwF+ [40] uniendo todas las capas específicas de cada tarea en una sola que se actualiza con una función de pérdida ajustada mediante fine-tunning a partir de un entrenamiento inicial. También en ese

mismo trabajo [40] plantean una combinación con EWC que resulta en un rendimiento similar denominada EWCLwF.

Otra rama es la de Aprendizaje incremental donde se busca una especie de entrenamiento online de datos cuando se dispone de su etiquetado. Sobre ella, [41] propone el uso de una función de pérdida que equilibre el coste de las predicciones contra una pérdida de destilación que corrige la discrepancia entre la respuesta de la red desde que ocurrieron las tareas pasadas hasta la actual. Supone costoso y además el rendimiento se degrada cada vez que se aplica esta nueva pérdida. Mientras que [42] es otra aproximación en este caso arquitectural que extiende sobre un modelo base de capas compartidas de pesos congelados añadiendo una capa extra de clasificación para cada tarea en función de las nuevas clases que contengan. También plantean otra versión en la que se modifica el porcentaje de la red que se congela mediante una matriz de similitud que se calcula durante el entrenamiento para permitir más elasticidad.

PackNet [43] agrupa múltiples tareas («Pack») secuencialmente en un modelo explotando la compresión en redes profundas [44]. Se reducen pesos que no afectan al entrenamiento (usando máscaras), reservándolos para nuevas tareas. También se utilizan el resto de pesos para continuar aprendiendo. Tiene como curiosidad que esta estrategia está diseñada sobre una red fija, por lo que hay una limitada capacidad de expansión y por tanto solo se pueden agrupar una cantidad de tareas. **PAE** (Packing and expanding) [45] ve esta limitación sobre la naturaleza estática de la arquitectura planteada en PackNet y la extiende añadiendo la posibilidad de aumentar y reducir el tamaño de la red. Hace que inicialmente sea más compacto y en largas secuencias más preciso. **PiggyBack** [46] desarrolla la idea del uso de los mismos pesos para diferentes tareas. Elimina la modificación de pesos de PackNet y PAE para enfocarse en el uso de máscaras (una por tarea) que se modifiquen con entrenamiento para las tareas. Sus autores afirman que el resultado es una arquitectura agnóstica a la naturaleza de las tareas, en cuanto a que la adición de tareas no afecta el rendimiento en ninguna otra y no hay un límite de tareas que pueden añadirse. Por otra parte sus resultados muestran que se moderadamente similar aunque algo mejor que PackNet incluso en ImageNet donde se podría pensar que debería haber enormes diferencias. La version más reciente de esta técnica es **MAS** [47], que evoluciona sus máscaras mediante técnicas no supervisadas en función de la importancia de cada peso, evitando que los pesos importantes se muevan demasiado (manteniendo plasticidad). **GMED** [48] extiende MAS al ámbito online eliminando los identificadores de tareas y sustituyéndolos por una pequeña memoria de revisión que se utiliza cuando se encuentra una reducción importante en el valor del gradiente. También **ESRIL** [49] extiende a MAS usando clusterings para construir una memoria de revisión óptima.

Por otra parte, los intentos de revisión buscan de cualquier manera apartarse del entrenamiento tradicional con estructuras que enmascaren el guardado de datos. Esto culmina con **SIGANN** [50]que utiliza un auto-encoder que aprende sobre el grupo de tareas y genera ejemplos con los que entrenar el modelo para recordar los ejemplos no disponibles. Otra solución también distante es **OvA-INN** [51], diseñado para el paradigma NC. En él entrenan una red neuronal Invertible (INN) para cada nueva clase encontrada. Estas redes extraen las características más relevantes para computar la probabilidad de que un ejemplo se encuentre en la clase. En test se predice cada ejemplo identificando cuales de todas las redes INN devuelve la mayor probabilidad. Más primitivamente y de manera previa, **Conditional Replay** [52] genera ejemplos con un muestreo a una distribución condicionada en la clase similar a [53].

Además de buscar solventar tareas de clasificación hay algunos aventureros que no se ven desanimados por el rendimiento del estado del arte y se embarcan en el área de segmentación. **ACE**[54] y [55] son los primeros. ACE, plantea el uso de un preprocesado de la imagen para ajustarlas al estilo de todas las tareas del stream. Sirve de normalización para facilitar el aprendizaje. A la hora de entrenar, se da como entrada el ejemplo procesado y sin procesar resultando en 2 predicciones sobre las que se derivan actualizaciones de pesos mediante divergencia de ambas. En [55] ataca el problema de forma completamente diferente. Utiliza una pareja de autoencoders sobre la que congela sobre la marcha las diferentes para reducir la variación de pesos.

4. Análisis de Soluciones

Las soluciones planteadas actualmente basan su éxito en la evaluación de ciertas métricas sobre unos benchmarks comúnmente utilizados. Dado que aún no se conoce ninguna manera teórica de determinar la efectividad de los métodos de aprendizaje continuo se requiere de análisis empíricos mediante pruebas de rendimiento.

El análisis de rendimiento de modelos supervisados offline conlleva únicamente evaluar el rendimiento sobre los datos de los que se requiere que rinda. En el caso de continual learning buscamos realizar una valoración de como la solución planteada puede llegar a rendir sobre cualquier conjunto de datos que cuantifique el nivel de completitud de los objetivos descritos en 1.3.

Un sistema de evaluación ideal recolectaría las métricas recogidas en 4.1 tras entrenar sobre uno de los benchmarks estándar como pueden ser 4.2. Esto facilitaría enormemente no solo la evaluación cruzada de propuestas si no también los logros aislados de la solución.

4.1. Métricas

La evaluación de las soluciones de CL requiere de una gran cantidad de información para poder determinar cuales de los objetivos planteados se han logrado o no cumplir. Por ello se han definido una serie de métricas que se alinean con los objetivos de CL mencionados en 1.3.

4.1.1. Transferencia de Información

Una de las métricas más relevantes es cuantificar como el aprendizaje de una tarea afecta al rendimiento sobre las demás. Permite ver la capacidad de la solución planteada para destilar información.

Dado que se entrena sobre T tareas y también se evalúa sobre esas mismas T se puede generar una matriz de precisión desde $[1, T]$. Donde $Acc_{i,j}$ es la precisión del modelo sobre el conjunto de test de la tarea j tras entrenar contra la tarea i y T es el número total de tareas.

	Test 1	**Test 2**	**Test 3**	...	**Test T**
Train 1	$Acc_{1,1}$	$Acc_{1,2}$	$Acc_{1,3}$	...	$Acc_{1,T}$
Train 2	$Acc_{2,1}$	$Acc_{2,2}$	$Acc_{2,3}$	...	$Acc_{2,T}$
Train 3	$Acc_{3,1}$	$Acc_{3,2}$	$Acc_{3,3}$	...	$Acc_{3,T}$
⋮	⋮	⋮	⋮	⋮	⋮
Train T	$Acc_{T,1}$	$Acc_{T,2}$	$Acc_{T,3}$	...	$Acc_{T,T}$

Cuadro 1: Matriz de precisiones en entrenamiento continuo.

Al contrario que un modelo tradicional sobre el que se evalúa la precisión sobre un conjunto de test para juzgar el rendimiento y viabilidad de una solución sobre datos reales, los objetivos de

Definida esta matriz es sencillo ver como el entrenamiento de una tarea afecta el olvido de las anteriores (BWT) o como una tarea permite destilar información sobre futuras sin haberlas visto (FWT). Ambas métricas pueden evaluarse a nivel de tarea, es decir, como aprender una tarea afecta a las anteriores o posteriores, o en general como la solución permite la transferencia de información. Este último suele ser más interesante ya que es mucha menos información (más concisa) y evalúa la solución en global en lugar de centrarse en las divisiones de los datos.

Positive Backward Transfer (BWT+) Mide la influencia de aprender nuevas tareas sobre el rendimiento de las tareas ya vistas (anteriores).

$$BWT^{+} = max\left(\frac{\sum_{i=2}^{T}\sum_{j=1}^{i-1} Acc_{i,j} - Acc_{j,j}}{T(T-1)/2}, 0\right) \tag{11}$$

Forward Transfer (FWT) Mide el posible impacto que tendrá el aprendizaje i sobre las futuras tareas.

$$FWT = \frac{\sum_{i=1}^{j-1}\sum_{j=1}^{T} Acc_{i,j}}{T(T-1)/2} \tag{12}$$

4.1.2. Precisión Media

Es la medida más útil para valorar el rendimiento del modelo sobre una tarea concreta. Evalúa el rendimiento del modelo sobre todas las tareas vistas previamente, en cualquier punto deseado del entrenamiento.

$$Acc_i = \frac{1}{i}\sum_{j=1}^{i} Acc_{i,j} \tag{13}$$

Si tomamos Acc_T, es decir, la precisión media en la última tarea, tenemos la medida de precisión habitual en un entrenamiento tradicional.

4.1.3. Olvido

El olvido 2.3 es el principal factor a optimizar. Esta métrica permite valorar numéricamente cuanta información se deja atrás en el proceso de entrenamiento al encontrar una nueva tarea con respecto a las anteriores por la solución planteada. No es representativo del rendimiento ya que a menor precisión conseguida por tarea, menor cota de olvido se consigue. Siendo 0 en el caso de no llegar nunca a aprender. El olvido sobre la tarea i se define como,

$$F_i = \frac{1}{i-1}\sum_{j=1}^{i-1} f_{i,j} \tag{14}$$

Siendo $f_{i,j}$ cuanto se ha olvidado sobre la tarea j tras ser entrenada en i.

$$f_{i,j} = max_{l\in\{1,\dots,i-1\}} Acc_{l,j} - Acc_{i,j} \ \forall j < i$$

El olvido medio da una imagen más general y se deriva del olvido por tarea.

$$F = \frac{1}{T}\sum_{i=1}^{T} F_i \tag{15}$$

4.1.4. Coste Computacional

Crecimiento del modelo En el caso de soluciones arquitecturales, se aumentan el número de pesos al incrementar la complejidad del modelo. Si no hubiera límite en el tamaño del modelo, los costes computacionales serían excesivos no cumpliendo los objetivos de una solución CL ideal.

$$ModelSize = min\left(1, \frac{\sum_{i=1}^{T}\frac{Mem(\theta_1)}{Mem(\theta_i)}}{T}\right) \tag{16}$$

$Mem(\theta_i)$ es el uso de memoria de los parámetros del modelo θ en el instante de haber entrenado la tarea número i.

Eficiencia de memoria de revisión Cuando se almacenan datos en las metodologías de revisión, se puede caer en la trampa de estar utilizando excesiva memoria sin conocerlo. El valor de

esta métrica está entre [0, 1], donde 0 es un uso de memoria de un entrenamiento tradicional (se guardan todos los datos) y 1 es ningún uso de memoria (entrenamiento online).

$$1 - min\left(1, \frac{\sum_{i=1}^{T} \frac{Mem(M^i)}{Mem(D)}}{T}\right) \tag{17}$$

Donde $Mem(M^i)$ es la cantidad de memoria usada para guarda ejemplos para la tarea i y $Mem(D)$ es la memoria que ocupa el dataset completo.

Eficiencia Computacional Todas las soluciones añaden una cierta latencia al proceso de entrenamiento y/o testeo, es requerido ya que se agregan mecanismos y rutinas. Pero demasiado proceso significa que la eficiencia se reduce. Al igual que la anterior métrica, el resultado varia entre $[0, 1]$ siendo 1 la mayor eficiencia y 0 la menor.

$$min\left(1, \frac{\sum_{i=1}^{T} \frac{POps(t_i)\epsilon}{Ops(t_i)}}{T}\right) \tag{18}$$

$POps(t_i)$ es la cantidad de operaciones requeridas para hacer una inferencia y una propagación hacia atrás sobre la tarea completa i. Mientras que $Ops(t_i)$ es el número total de operaciones para aprender la tarea i.

4.2. Benchmarks

La evaluación de rendimiento en modelos tradicional consiste en la división de los datos en un conjunto de entrenamiento y otro de validación (y test) sobre el que se mide la precisión para determinar su calidad. Tiene un bajo número de variables, (1) Afinación de los Hiperparámetros (fine-tunning) para el entrenamiento (2) Selección del Dataset (3) División del dataset en entrenamiento-test.

Las soluciones CL construyen a partir de estas variables y añaden dos más, (4) División del dataset en bloques o estructura de stream (5) Formato de los bloques de datos (entre los 3 tipos descritos anteriormente 3.3).

De entre estas variables presentadas se puede hacer un cribado. Debido a que los hiper-parámetros son altamente dependientes del modelo seleccionado y solo se requiere una configuración óptima podemos descartarlos para el análisis. De igual forma, la división de dataset es un factor ya estudiado siendo muy común una división 70-30 % para asegurar una correcta representación probabilística de los datos (varia en función del conjunto de datos). En CL está división es realizada de varias formas, un 70-30 % antes del reparto en tareas sin llegar nunca a dividir el conjunto de test, una división tras haber hecho el reparto o no utilizar test y solo evaluar sobre train dado que no se busca la representatividad de los datos si no la capacidad de retener lo aprendido.

Los **benchmarks** son una serie de pruebas comunes que tratan de evaluar de manera consistente entre publicaciones el rendimiento de las soluciones planteadas. Surgen con la intención de normalizar la evaluación de soluciones CL que facilite la generación de resultados empíricos y permita comparaciones entre planteamientos. Hay 3 parámetros que definen un benchmark y determinan la prueba a realizar.

1. Dataset.
2. División del dataset (NI, NC, NIC).
3. Formato de los bloques de datos (SIT, MT).

4.2.1. Selección de Dataset

En cuanto a la selección del dataset, destacar que al encontrarse el SoTA en una fase de exploración, se buscan conjuntos de datos representativos que puedan mostrar las posibles carencias

del método a testear. Por ello aún habiendo partido inicialmente de utilizar MNIST propuesto en [56], un conjunto de datos de clasificación de dígitos, en los trabajos más recientes se prescinde de su uso ya que como se menciona en [57] es demasiado sencillo, pequeño y muy poco representativo de datos reales para servir como referencia. Por otra parte, se han intentado utilizar otros como ImageNet que es el benchmark por excelencia en clasificación tradicional, pero resulta demasiado complejo para las soluciones actuales por lo que no

Los principales datasets utilizados son los conjuntos de datos comunes de clasificación en visión por computador.

MNIST [56] es uno de los primeros datasets utilizados en visión por computador por su simplicidad y lo sencillo que resulta su manejo. Surge como una partición de la base de datos de NIST. Esta formado por 60 000 imágenes de 28x28 píxeles en escala de grises de dígitos del 0 al 9 escritos a mano. La división train-test resulta en 50 000 y 10 000 imágenes respectivamente. Es conocido por ser lo mínimo sobre lo que probar una arquitectura de clasificación pero no es representativo de problemas de CV reales.

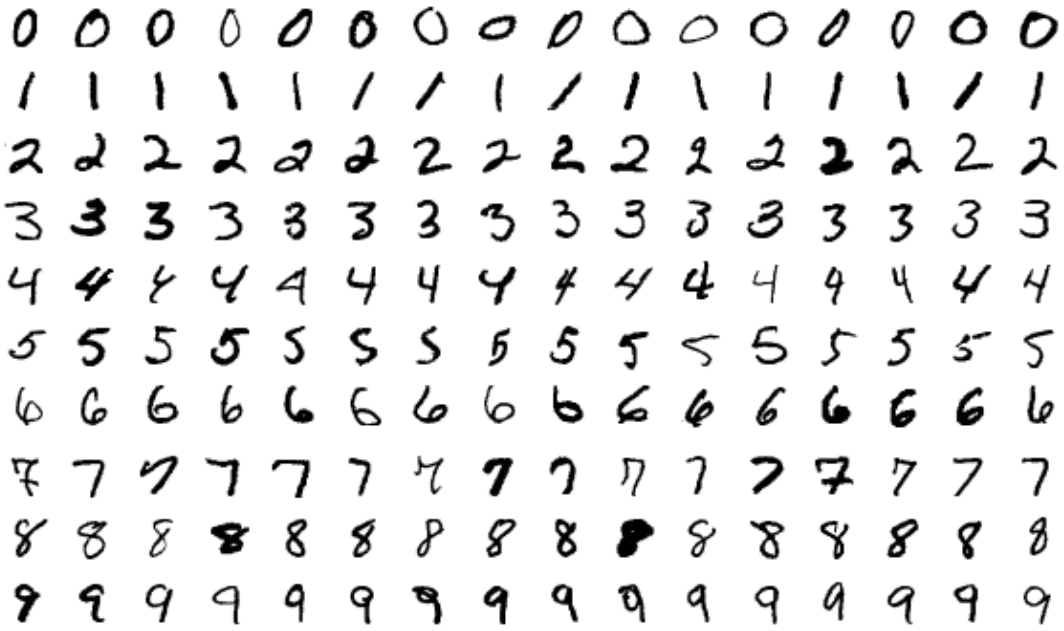

Figura 17: MNIST Dataset.

Como alternativa se presentan 2 conjuntos de datos que tratan de aumentar la complejidad, «Permuted MNIST» [58] y «Rotated MNIST». **Permuted MNIST** es una variante de MNIST con el mismo número de imágenes y mismo origen de las mismas. Varia en que se genera a partir de este pero permutando ciertos píxeles de cada dígito. Esta transformación es la misma para todas las imágenes de la misma clase y se selecciona de manera aleatoria para cada una. **Rotated MNIST** es similar, en cuanto a que se genera mediante transformaciones aleatorias. Se toman bloques de n en n imágenes (4 por defecto) y se rotan un cierto ángulo entre $[-\pi, \pi]$ radianes.

Fashion MNIST [59] es una evolución de MNIST que plantea un problema similar con imágenes del mismo formato y el mismo número de clases pero esta vez de ropa en lugar de dígitos. Esta similitud permite intercambiar entre ambos conjuntos sin modificar la estructura, permitiendo evaluar sobre otro dataset sin demasiado esfuerzo. En la práctica resulta un conjunto más complejo sin modificar la estructura. Las clases son `[T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot]` ver 18. También existe otra variante que utiliza caracteres en lugar de cifras pero no es muy usado por ser demasiado similar a MNIST y bastante menos conocido.

Figura 18: FashionMNIST Dataset. Esta figura ha sido ampliada para facilitar la visualización de las clases, en realidad las imágenes de 28x28 son muy complejas de distinguir.

Cifar-10 junto a **Cifar-100** [60] son un subconjunto de el dataset ImageNet [61]. Están formados por 60 000 imágenes en color de 32x32 divididas en 10 y 100 clases respectivamente. La división en Cifar-100 es jerárquica, es decir, se compone de 20 superclases con 100 heredando de éstas, ver 2, donde cada clase contiene solo 600 ejemplos, 500 de entrenamiento y 100 para test.

Super clases	**Clases**
aquatic mammals	beaver, dolphin, otter, seal, whale
fish	aquarium fish, flatfish, ray, shark, trout
flowers	orchids, poppies, roses, sunflowers, tulips
food containers	bottles, bowls, cans, cups, plates
fruit and vegetables	apples, mushrooms, oranges, pears, sweet peppers
household electrical devices	clock, computer keyboard, lamp, telephone, television
household furniture	bed, chair, couch, table, wardrobe
insects	bee, beetle, butterfly, caterpillar, cockroach
large carnivores	bear, leopard, lion, tiger, wolf
large man-made outdoor things	bridge, castle, house, road, skyscraper
large natural outdoor scenes	cloud, forest, mountain, plain, sea
large omnivores and herbivores	camel, cattle, chimpanzee, elephant, kangaroo
medium-sized mammals	fox, porcupine, possum, raccoon, skunk
non-insect invertebrates	crab, lobster, snail, spider, worm
people	baby, boy, girl, man, woman
reptiles	crocodile, dinosaur, lizard, snake, turtle
small mammals	hamster, mouse, rabbit, shrew, squirrel
trees	maple, oak, palm, pine, willow
vehicles 1	bicycle, bus, motorcycle, pickup truck, train
vehicles 2	lawn-mower, rocket, streetcar, tank, tractor

Cuadro 2: Clases Cifar-100

La versión de 10 clases en cambio, son agrupaciones de las superclases mencionadas anteriormente con 6000 imágenes cada una. El reparto train-test es de 50 000 a 10 000. Este conjunto es bastante representativo de problemas simples de clasificación en CV. Aunque la resolución sea algo baja, el número de clases es bastante adecuado para experimentación.

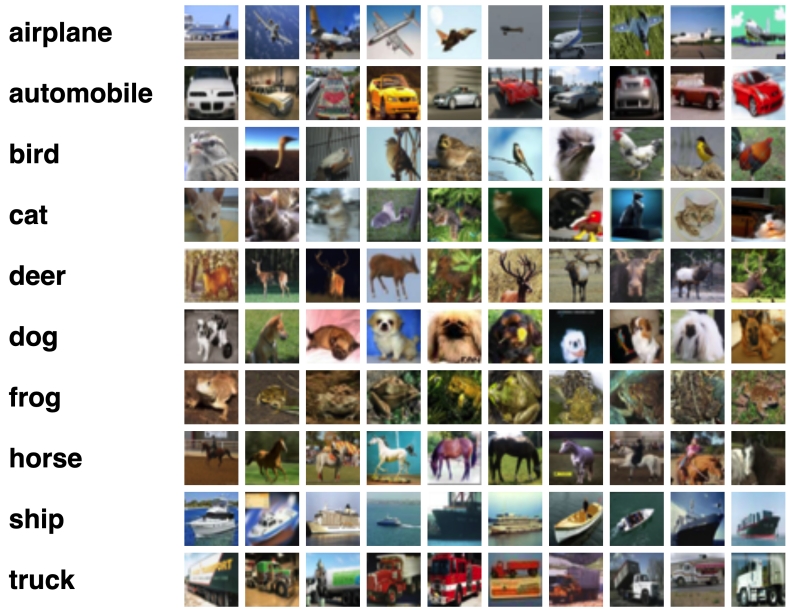

Figura 19: Clases Cifar-10.

MiniImagenet [62] es un subconjunto de ImageNet (al igual que Cifar). Debido al enorme tamaño y complejidad que supone ImageNet, se ha reducido a una colección de 60 000 imágenes a color de 84x84 divididas en 100 clases con 600 ejemplos cada una. El argumento para el tamaño de esta selección es que resulta más complejo que estándares como CIFAR10 mientras que ocupa la cantidad justa para entrar en memoria en la mayoría de sistemas. Haciendo que el prototipado y experimentación con el sea conveniente y rápido.

CORe50 [30] Dataset de clasificación de imágenes. Diseñado con el propósito de evaluar soluciones de aprendizaje continuo. Esta formado por unas 3300 imágenes de diferentes objetos suspendidos sobre un entorno, ver 24. Son de tamaño 128x128 más RGB, divididas en 10 o 50 clases. Similar a Cifar100 pero en este caso intencionalmente, las clases se encuentran recogidas en superclases con una relación de similaridad.

Clases	**Objeto al que representan**
`[o1, ..., o5]`	plug adapters
`[o6, ..., o10]`	mobile phones
`[o11, ..., o15]`	scissors
`[o16, ..., o20]`	light bulbs
`[o21, ..., o25]`	cans
`[o26, ..., o30]`	glasses
`[o31, ..., o35]`	balls
`[o36, ..., o40]`	markers
`[o41, ..., o45]`	cups
`[o46, ..., o50]`	remote controls

Cuadro 3: Clases CORe50.

4.2.2. Benchmarks Comunes

Dataset	NI	NC	NIC	MT	SI
Split MNIST (Fashion and Digits)		✓		✓	✓
Rotarion MNIST	✓				
Permutation MNIST	✓				
Split CIFAR10/100		✓			
ILSVRC		✓		✓	
CUB200	✓				
CORe50	✓	✓	✓		
Split MiniImageNet		✓			
CLEAR Benchmark		✓		✓	

Cuadro 4: Manejo de datos de los principales benchmarks presentados.

Split MNIST es un benchmark sobre MNIST y Fashion MNIST que los agrupa de 2 maneras principalmente [31]. La primera es una agrupación NC-MT que une las clases de 2 en 2 resultando en 5 tareas total, por ejemplo (0,1) en la primera tarea, (2,3) la siguiente, etc, en el caso de MNIST y similarmente para FashionMNIST. La siguiente es la evaluación clase a clase de todo el dataset, NC-SIT.

iCIFAR-100 Introducido en [36] se trata de una división de Cifar-100 en grupos de tareas de 2, 5, 10, 20 o 50 clases para una agrupación NC-MT. El caso concreto de separar en tareas de 5 clases se denomina Split CIFAR-100 que se construye dividiendo Cifar-100 en 20 tareas de 5 clases disjuntas cada una. Cada tarea incluye 2500 imágenes (3x32x32) para entrenamiento y 500 para test.

Split MiniImageNet, este subconjunto de ImageNet divide las 100 clases en 20 grupos con 5 clases cada una, para un total de 2500 imágenes para entrenamiento (3x84x84) y 500 para test.

Al igual que iCifar100 **ILSVRC** fue presentado en [36]. Es un subconjunto de ImageNet llamado ILSVRC2012 que se agrupa de 2 formas. La primera denominada iILSVRC-small que utiliza 100 clases y las divide en tareas de 10 clases cada una. Y iILSVRC-full que utiliza todas las 1000 clases y las agrupa de 100 en 100 para un total de 10 tareas.

Split CUB-200 es una versión iterativa de el dataset Caltech-UCSD Birds [63].

CORe50 inicialmente se planteó con 2 versiones, NC y NI, pero dada su popularidad se ha extendido para cubrir todos los casos de uso existentes. Estos supone, el caso NC donde cada tarea consta de 5 clases concretas de 5 en 5 en el orden indicado (0-50) en la web del dataset. NI tiene un orden bastante aleatorio. Y la versión NIC dispone de 4 versiones, NIC, NICv2_79, NICv2_196 y NICv2_391

Protocol	# batches	Initial batch		Incremental Batches	
		# Classes	# Images	# Classes	# Images
NIC	79	10	3,000	5	1,500
NICv2-79	79	10	3,000	5	1,500
NICv2-196	196	10	3,000	2	600
NICv2-391	391	10	3,000	1	300

Figura 20: Variantes NIC de CORe50

CLEAR [64] es un benchmark creado exclusivamente con el propósito de estudiar soluciones CL. Creado recientemente, contiene unas 7.8M de imágenes extraídas de YFCC100M, la colección de imágenes públicas de mayor tamaño. Se encuentran ordenadas en un stream de datos desde 2004 hasta 2014 y agrupadas en 11 clases con 700k fotos cada una. De la colección completa mencionada, se ha creado un subconjunto etiquetado con proporciones más compactas para facilitar su uso. Consiste en 10 clases con 300 imágenes por clase en formato NC-MT.

5. Conclusiones

En esta memoria se ha planteado las motivaciones para continuar investigando el aprendizaje continuo, las bases teóricas sobre las que se basan estas motivaciones, una clasificación del estado del arte dando unas pinceladas sobre el mismo y finalmente una forma de evaluar las propuestas encontradas.

Tras ello como conclusión, al contrario que el problema de aprendizaje en redes neuronales del que se conocía perfectamente y se pudo encontrar una solución matemática óptima tal que con una función multivariable de caja negra se pueda representar un espacio de soluciones suficiente para representar datos multidimensionales. Continual Learning es un paradigma de mucha más envergadura con muy poca consistencia en sus bases. Dispone de un crisol de ideas y planteamientos, muchas publicaciones consisten en tratar de acotar el problema mientras que otras toman alguno de los diferentes paradigmas planteados para evolucionar sus soluciones.

Idealmente se dispondría de un problema perfectamente definido, reduciendo el trabajo de plantear una solución y facilitando el avance del SoTA. En cuanto a el futuro de propuestas, dado que aún no existe la base teórica requerida toda investigación se basa en el prueba y error sobre benchmarks como destaco en el apartado de evaluaciones.

5.1. Evaluación de soluciones CL

Parte de esta consistencia buscada viene de la comparación entre métodos. Actualmente es muy complejo realizar comparaciones y requiere mucha interpretación sobre los datos presentados. Las pruebas resultadas son demasiado heterogéneas y las comparaciones son muy complejas en incluso en algunos casos imposibles a partir de los resultados dados por los autores. En esta memoria se ha descrito las posibles métricas, datos y su formato a utilizar para conseguir una base sólida sobre la que evaluar una solución propuesta. Además, utilizando el framework open-source Avalanche [65] se ha desarrollado unos «logger» (herramienta de recogida de resultados) que se puede añadir a una solución CL para conseguir la matriz de precisiones y otras utilidades relacionadas con su evaluación. El código está disponible en mi repositorio. También se programaron una serie de pruebas para los 3 modelos descritos en detalle en esta memoria. Lamentablemente los resultados no son los esperados, pues el proyecto aún se encuentra en una fase muy joven, si desean conseguirse el código está disponible.

5.2. Trabajo futuro

En el futuro sería interesante la implementación de un sistema de evaluación automatizado de propuestas. Junto con la realización de pruebas sobre métodos existentes. Además, dado que CL está sufriendo un resurgimiento y una enorme cantidad de propuestas, convendría realizar un análisis más profundo sobre las soluciones que no se han desarrollado del todo en esta memoria y seguir su trayectoria.

Referencias

[1] Robert Geirhos, David HJ Janssen, Heiko H Schütt, Jonas Rauber, Matthias Bethge, and Felix A Wichmann. Comparing deep neural networks against humans: object recognition when the signal gets weaker. *arXiv preprint arXiv:1706.06969*, 2017.

[2] Antoine Buetti-Dinh, Vanni Galli, Sören Bellenberg, Olga Ilie, Malte Herold, Stephan Christel, Mariia Boretska, Igor V Pivkin, Paul Wilmes, Wolfgang Sand, et al. Deep neural networks outperform human expert's capacity in characterizing bioleaching bacterial biofilm composition. *Biotechnology Reports*, 22:e00321, 2019.

[3] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. *CoRR*, abs/2104.00298, 2021.

[4] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 2020.

[5] Tong Yu and Hong Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

[6] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[7] Fabio Henrique Kiyoiti dos Santos Tanaka and Claus Aranha. Data augmentation using gans. *arXiv preprint arXiv:1904.09135*, 2019.

[8] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.

[9] Sebastian Thrun and Tom M. Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1):25–46, 1995. The Biology and Technology of Intelligent Autonomous Agents.

[10] Zhiyuan Chen, Nianzu Ma, and Bing Liu. Lifelong learning for sentiment classification. *arXiv preprint arXiv:1801.02808*, 2018.

[11] Bing Liu. Lifelong machine learning: a paradigm for continuous learning. *Frontiers of Computer Science*, 11(3):359–361, 2017.

[12] Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

[13] Abien Fred Agarap. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018.

[14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[15] Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-aware minimization for efficiently improving generalization. *CoRR*, abs/2010.01412, 2020.

[16] Christopher W Lynn and Danielle S Bassett. How humans learn and represent networks. *Proceedings of the National Academy of Sciences*, 117(47):29407–29415, 2020.

[17] Stevo Bozinovski and Ante Fulgosi. The influence of pattern similarity and transfer learning upon training of a base perceptron b2. In *Proceedings of Symposium Informatica*, volume 3, pages 121–126, 1976.

[18] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *CoRR*, abs/1911.02685, 2019.

[19] Raia Hadsell, Dushyant Rao, Andrei A Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in cognitive sciences*, 24(12):1028–1040, 2020.

[20] Michael McCloskey and Neal J Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In *Psychology of learning and motivation*, volume 24, pages 109–165. Elsevier, 1989.

[21] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146, 1995.

[22] Ameya Prabhu, Philip HS Torr, and Puneet K Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *European conference on computer vision*, pages 524–540. Springer, 2020.

[23] Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, Yuxin Peng, and Zheng Zhang. Error-driven incremental learning in deep convolutional neural network for large-scale image classification. In *ACM Multimedia*, November 2014.

[24] Matthias De Lange and Tinne Tuytelaars. Continual prototype evolution: Learning online from non-stationary data streams. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8250–8259, October 2021.

[25] Zheda Mai, Ruiwen Li, Jihwan Jeong, David Quispe, Hyunwoo Kim, and Scott Sanner. Online continual learning in image classification: An empirical survey. *CoRR*, abs/2101.10423, 2021.

[26] Davide Maltoni and Vincenzo Lomonaco. Continuous learning in single-incremental-task scenarios. *CoRR*, abs/1806.08568, 2018.

[27] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

[28] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

[29] Vincenzo Lomonaco, Davide Maltoni, and Lorenzo Pellegrini. Rehearsal-free continual learning over small non-iid batches. In *CVPR Workshops*, volume 1, page 3, 2020.

[30] Vincenzo Lomonaco and Davide Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 17–26. PMLR, 13–15 Nov 2017.

[31] Friedemann Zenke, Ben Poole, and Surya Ganguli. Improved multitask learning through synaptic intelligence. *CoRR*, abs/1703.04200, 2017.

[32] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[33] Bernd Fritzke. A growing neural gas network learns topologies. *Advances in neural information processing systems*, 7, 1994.

[34] Stephen Marsland, Jonathan Shapiro, and Ulrich Nehmzow. A self-organising network that grows when required. *Neural Networks*, 15(8):1041–1058, 2002.

[35] Zhizhong Li and Derek Hoiem. Learning without forgetting. *CoRR*, abs/1606.09282, 2016.

[36] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. *CoRR*, abs/1611.07725, 2016.

[37] Raffaello Camoriano, Giulia Pasquale, Carlo Ciliberto, Lorenzo Natale, Lorenzo Rosasco, and Giorgio Metta. Incremental robot learning of new objects with fixed update time. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3207–3214. IEEE, 2017.

[38] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continuum learning. *CoRR*, abs/1706.08840, 2017.

[39] Arslan Chaudhry, Marc'Aurelio Ranzato, Marcus Rohrbach, and Mohamed Elhoseiny. Efficient lifelong learning with A-GEM. *CoRR*, abs/1812.00420, 2018.

[40] Hyo-Eun Kim, Seungwook Kim, and Jaehwan Lee. Keep and learn: Continual learning by constraining the latent space for knowledge preservation in neural networks. *CoRR*, abs/1805.10784, 2018.

[41] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. *CoRR*, abs/1708.06977, 2017.

[42] Syed Shakib Sarwar, Aayush Ankit, and Kaushik Roy. Incremental learning in deep convolutional neural networks using partial network sharing. *CoRR*, abs/1712.02719, 2017.

[43] Arun Mallya and Svetlana Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. *CoRR*, abs/1711.05769, 2017.

[44] James O'Neill. An overview of neural network compression. *CoRR*, abs/2006.03669, 2020.

[45] Steven Hung, Jia-Hong Lee, Timmy Wan, Chein-Hung Chen, Yi-Ming Chan, and Chu-Song Chen. Increasingly packing multiple facial-informatics modules in a unified deep-learning model via lifelong learning. pages 339–343, 06 2019.

[46] Arun Mallya and Svetlana Lazebnik. Piggyback: Adding multiple tasks to a single, fixed network by learning to mask. *CoRR*, abs/1801.06519, 2018.

[47] Rahaf Aljundi, Francesca Babiloni, Mohamed Elhoseiny, Marcus Rohrbach, and Tinne Tuytelaars. Memory aware synapses: Learning what (not) to forget. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 139–154, 2018.

[48] Rahaf Aljundi, Klaas Kelchtermans, and Tinne Tuytelaars. Task-free continual learning. *CoRR*, abs/1812.03596, 2018.

[49] Lei Guo, Gang Xie, Xinying Xu, and Jinchang Ren. Exemplar-supported representation for effective class-incremental learning. *IEEE Access*, 8:51276–51284, 2020.

[50] Diego Mellado, Carolina Saavedra, Steren Chabert, Romina Torres, and Rodrigo Salas. Self-improving generative artificial neural network for pseudorehearsal incremental class learning. *Algorithms*, 12(10), 2019.

[51] Guillaume Hocquet, Olivier Bichler, and Damien Querlioz. Ova-inn: Continual learning with invertible neural networks. In *2020 international joint conference on neural networks (IJCNN)*, pages 1–7. IEEE, 2020.

[52] Timothée Lesort, Alexander Gepperth, Andrei Stoian, and David Filliat. Marginal replay vs conditional replay for continual learning. *CoRR*, abs/1810.12069, 2018.

[53] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *CoRR*, abs/1705.08690, 2017.

[54] Zuxuan Wu, Xin Wang, Joseph E. Gonzalez, Tom Goldstein, and Larry S. Davis. ACE: adapting to changing environments for semantic segmentation. *CoRR*, abs/1904.06268, 2019.

[55] Umberto Michieli and Pietro Zanuttigh. Incremental learning techniques for semantic segmentation. *CoRR*, abs/1907.13372, 2019.

[56] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[57] Matthias De Lange, Rahaf Aljundi, Marc Masana, Sarah Parisot, Xu Jia, Aleš Leonardis, Gregory Slabaugh, and Tinne Tuytelaars. A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385, 2021.

[58] Rupesh K Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber. Compete to compute. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[59] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[60] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[61] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: a large-scale hierarchical image database. pages 248–255, 06 2009.

[62] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *CoRR*, abs/1606.04080, 2016.

[63] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[64] Zhiqiu Lin, Jia Shi, Deepak Pathak, and Deva Ramanan. The CLEAR benchmark: Continual learning on real-world imagery. *CoRR*, abs/2201.06289, 2022.

[65] *Avalanche: an End-to-End Library for Continual Learning*, 2nd Continual Learning in Computer Vision Workshop. Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2021.

Anexo

Revisión sistemática

Fase 1 Estudios obtenidos utilizando el criterio de búsqueda mencionado dentro de Scopus. 103 artículos resultantes. `(TITLE-ABS-KEY ((«continual learning» OR «adaptative learning» OR «lifelong learning» OR «catastrophic forgetting» OR «catastrophical forgetting») AND («machine learning» OR «deep learning» OR «neural network» OR «Artificial intelligence»)) AND LANGUAGE («English»)) AND PUBYEAR >2009 AND PUBYEAR <2021 AND (LIMIT-TO (SUBJAREA , «COMP»))`

Fase 2 Se revisan los títulos y resúmenes de los trabajos obtenidos para descartar aquellos que no apliquen y los menos relevantes. 74 artículos resultantes.

Criterios de inclusión (CI)

CI1. Artículos con al menos 1 cita.

CI2. Artículos open-access.

CI3. Basados en sistemas con redes neuronales.

Criterios de exclusión (CE)

CE1. Artículos aplicados a un caso práctico poco relevante o sin interés para el estudio.

CE2. Artículos que no se puedan replicar con facilidad.

CE3. Artículos de opinión o de resumen.

CE4. Artículos duplicados, manteniendo la versión más reciente del mismo.

Fase 3 Se estudian en profundidad los diferentes artículos obtenidos en la Fase 2, aplicando los criterios de inclusión y exclusión establecidos. 58 artículos + 8 por snowball.

5.3. Comparación de métodos planteados

Como ya planteado en esta memoria, la comparación de resultados no es para nada trivial. En este caso podemos ver por ejemplo que tanto en GDumb como en CoPE se evalúa también MIR sobre MNIST pero aún siendo el mismo benchmark y solución los resultados varían muy considerablemente $93{,}20 \pm 0{,}36$ en el caso de CoPE mientras que en los resultados de GDumb $84{,}9 \pm 1{,}7$. El mismo caso se da con los resultados sobre GEM.

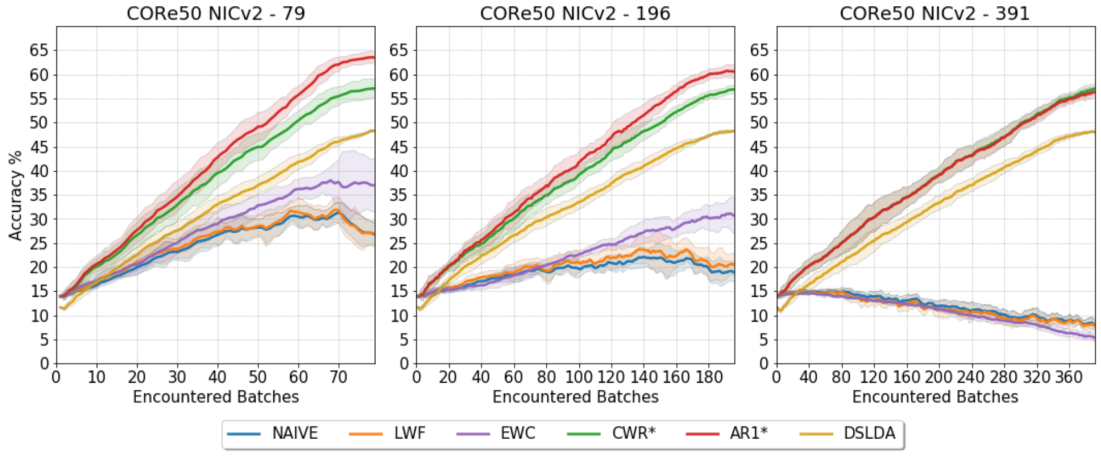

Figura 21: Resultados empíricos de AR1* sobre CORe50.

	Split-MNIST	**Split-CIFAR10**	**Split-CIFAR100**
iid-offline	98.44 ± 0.02	83.02 ± 0.60	50.28 ± 0.66
iid-online	96.57 ± 0.14	62.31 ± 1.67	20.10 ± 0.90
finetune	19.75 ± 0.05	18.55 ± 0.34	3.53 ± 0.04
GEM	93.25 ± 0.36	24.13 ± 2.46	11.12 ± 2.48
iCARL	83.95 ± 0.21	37.32 ± 2.66	10.80 ± 0.37
CURL [36]	92.59 ± 0.66	–	–
DN-CPM [29]	93.23 ± 0.09	45.21 ± 0.18	20.10 ± 0.12
reservoir	92.16 ± 0.75	42.48 ± 3.04	19.57 ± 1.79
MIR	93.20 ± 0.36	42.80 ± 2.22	20.00 ± 0.57
GSS	92.47 ± 0.92	38.45 ± 1.41	13.10 ± 0.94
CoPE-CE	91.77 ± 0.87	39.73 ± 2.26	18.33 ± 1.52
CoPE (ours)	$\mathbf{93.94 \pm 0.20}$	$\mathbf{48.92 \pm 1.32}$	$\mathbf{21.62 \pm 0.69}$

Figura 22: Resultados empíricos de CoPE.

Method k	MNIST (500)	CIFAR-10 (500)
Fine tuning	18.8 ± 0.6	18.5 ± 0.2
AGEM [36]	29.0 ± 5.3	18.5 ± 0.6
BGD [48]	13.5 ± 5.1	18.2 ± 0.5
GEM [7]	87.2 ± 1.3	20.1 ± 1.4
GSS-Greedy [37]	84.2 ± 2.6	28.0 ± 1.3
HAL [35]	77.9 ± 4.2	32.1 ± 1.5
ER [44]	81.0 ± 2.3	33.3 ± 1.5
MIR [11]	84.9 ± 1.7	34.5 ± 2.0
GMED (ER) [12]	82.7 ± 2.1	35.0 ± 1.5
GMED (MIR) [12]	87.9 ± 1.1	35.5 ± 1.9
GDumb (Ours)	$\mathbf{91.9 \pm 0.5}$	$\mathbf{45.8 \pm 0.9}$

(A2)

Method	MNIST		CIFAR10	
	Memory	Accuracy	Memory	Accuracy
Finetune	0	18.8 ± 0.5	0	15.0 ± 3.1
GEN [28]	4.58	79.3 ± 0.6	34.5	15.3 ± 0.5
GEN-MIR [11]	4.31	82.1 ± 0.3	38.0	15.3 ± 1.2
LwF [3]	1.91	33.3 ± 2.5	4.38	19.2 ± 0.3
ADI [47]	1.91	55.4 ± 2.6	4.38	24.8 ± 0.9
ARM [41]	1.91	56.2 ± 3.5	4.38	26.4 ± 1.2
ER [44]	0.39	83.2 ± 1.9	3.07	41.3 ± 1.9
ER-MIR [11]	0.39	85.6 ± 2.0	3.07	47.6 ± 1.1
iCarl [4] (5 iter)	-	-	3.07	32.4 ± 2.1
GEM [7]	0.39	86.3 ± 0.1	3.07	17.5 ± 1.6
GDumb (ours)	**0.39**	**91.9 ± 0.5**	**3.07**	**61.3 ± 1.7**

(A3)

Figura 23: Resultados pruebas GDumb.

Figuras Extra

Figura 24: Ejemplos de todas las clases en CORe50.

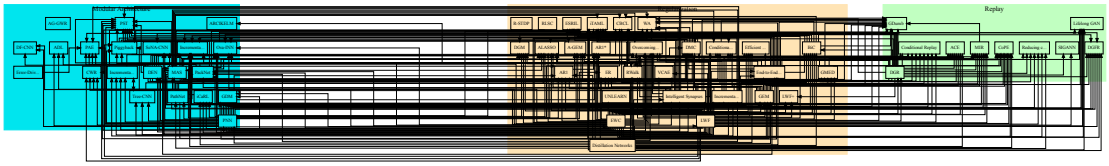

Figura 25: Grafo de referencias cruzadas de los artículos resultantes de la revisión sistemática con una clasificación primitiva.