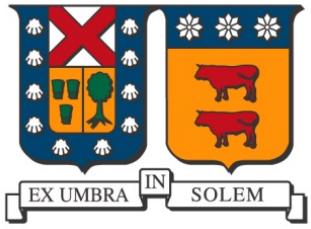
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“PROPUESTA DE ESTRATEGIAS PARA REDUCIR EL FENÓMENO DE OLVIDO CATASTRÓFICO EN MODELOS DE CLASIFICACIÓN CON APRENDIZAJE CONTINUO”

JESÚS VICENTE OYANEDEL ESPINOZA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Claudio Torres
Profesor Correferente: Elizabeth Montero, Raquel Pezoa

Enero - 2024

DEDICATORIA

Este trabajo va dedicado a mis padres, familiares y toda la gente que siempre ha creído en mi. Sin ustedes esto no sería posible.

AGRADECIMIENTOS

En primer lugar, quiero agradecer a mis padres y familia por ser un apoyo incondicional a lo largo de mi vida, y por extensión, de este trabajo de título, por ser conscientes del esfuerzo que me ha tomado llegar hasta este punto. También quiero agradecer a Anllerly Lazcano, mi pareja quien ha sido fundamental durante los últimos 7 años de mi vida, ganándose un lugar irreemplazable en mi vida dándome el apoyo que todo ser humano necesita.

Del mismo modo, quiero agradecer a mis compañeros de carrera Diego Barboza y Diego Quezada, quienes han sido de mucho utilidad para la culminación de este trabajo, fruto de meses realmente estresantes de esfuerzo, con quienes compartí mucho tiempo en clases, trabajos y avanzando este propio trabajo. José Quezada, Jonathan Pozo (*F* como dicen los lolos), Joaquín Rojas, Braulio Fuentes, Ignacia Delaigue, Benjamín Sánchez y todo el *team Chipleki*, quienes hicieron mi estancia en la Universidad una experiencia más amena.

Quiero agradecer a mi profesor guia, Claudio Torres, quien me recibió con los brazos abiertos luego de haber tenido una mala experiencia con mi trabajo de título anterior, resultante en el abandono del mismo. Sin él, ni su metodología de avances, seguramente no hubiese terminado este trabajo. Así mismo, toda persona involucrada en mi trabajo de título anterior también forman parte de este agradecimiento, pues la experiencia vivida cambió mi forma de percibir la vida, mi entorno, mis responsabilidades, me ayudó a crecer y madurar.

Los profesores y profesoras que fueron parte de mi formación académica, educación básica, media y universitaria les agradezco haberme formado como persona.

Todas las personas mencionadas, y las que no, son parte fundamental de este proceso, quienes han dejado huella en mí y pueden sentirse orgullosos y orgullosas de ser parte de mi historia y precursores de quien soy hoy, y probablemente de quién seré en el futuro.

RESUMEN

Resumen— La interferencia catastrófica (*catastrophic forgetting*) es un fenómeno presente en el entrenamiento de redes neuronales al realizar un entrenamiento continuo. En la medida que se entrenan nuevas tareas, la red empieza a olvidar las tareas previamente aprendidas. Este trabajo busca comparar distintas soluciones del estado del arte y realizar una serie de propuestas con el fin de apaciguar el efecto que tiene el olvido catastrófico en el entrenamiento secuencial de tareas.

Palabras Clave— Aprendizaje de máquinas, Aprendizaje continuo, Olvido catatrófico

ABSTRACT

Abstract— Catastrophic interference is a phenomenon present in the training of neural networks when continuous training takes place. As new tasks are learned, the network starts to forget previously learned tasks. This work compares state-of-the-art solutions and proposes several methods to mitigate the catastrophic forgetting effect in sequential task learning.

Keywords— *Machine Learning*, *Continual Learning*, *Catastrophic Forgetting*

GLOSARIO

$\mathcal{L}$: Función de pérdida general.

$\mathcal{L}_c$: Función de pérdida por clase. *Multi class cross entropy*.

$\boldsymbol{\theta}$: Parámetros / pesos actuales de la red. $\boldsymbol{\theta} \in \mathbb{R}^P$.

$\boldsymbol{\theta}^*_A$: Parámetros de la red en la tarea anterior. $\boldsymbol{\theta}^*_A \in \mathbb{R}^P$.

$\|\cdot\|_p$: Norma p.

$\odot$: Producto hadamard.

f: Modelo.

C: Número total de clases.

P: Número total de parámetros.

AI: *Artificial Intelligence*. Inteligencia artificial.

ML: *Machine Learning*. Aprendizaje de máquina.

CL: *Continual Learning*. Aprendizaje continuo.

ANN: *Artificial Neural Network*. Red neuronal artificial.

Naive: Ingenuo. Estrategia que no implementa técnicas para evitar el olvido catastrófico.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

A lo largo de los últimos años, hemos sido testigos de una creciente popularidad en el ámbito de la inteligencia artificial. Esta ha intervenido en nuestra vida de forma sorpresiva para las personas menos atentas al crecimiento de la tecnología, sin embargo, para el resto de las personas, la inteligencia artificial es el paso natural que debe seguir la humanidad en su evolución. Esta alza en interés y reconocimiento se ha intensificado, en gran parte, gracias a la presentación y difusión de modelos avanzados de procesamiento de lenguaje natural, conocidos como Large Language Models. Entre los más destacados de estos modelos encontramos a *GPT-4* [Cha, 2024b], modelo desarrollado por la empresa *OpenAI*, a *Llama* [Lla, 2024], una creación de *Meta* (previamente conocida como *Facebook*), y a *Bard* [Bar, 2024], producto de *Google*. Estas innovaciones no sólo han mostrado una impresionante capacidad técnica, sino que también han cambiado radicalmente la manera en la que las personas se relacionan e interactúan con la tecnología y los sistemas informáticos. No obstante, es importante subrayar que la inteligencia artificial no se limita únicamente a estos modelos. En realidad, este campo abarca una diversidad de áreas de investigación y aplicaciones.

El campo del aprendizaje automático se centra en el diseño y desarrollo de algoritmos que permiten a las máquinas aprender de grandes conjuntos de datos. La meta es que estas máquinas, a través de un proceso de entrenamiento, puedan identificar patrones y regularidades en los datos para luego hacer predicciones con nueva información que se les presente. Esta área del conocimiento se diversifica en varios tipos de aprendizaje, y a grandes rasgos, podemos clasificarlos en tres categorías principales (algunas veces cuatro) [Burkov, 2019].

La primera categoría es el *aprendizaje supervisado*. Este enfoque se basa en proporcionar a la máquina un conjunto de datos donde cada entrada viene acompañada de una etiqueta o valor específico. La máquina, utilizando estos datos, es entrenada para hacer predicciones o clasificaciones sobre información nueva que no haya sido vista durante su fase de entrenamiento. Como ilustración de este concepto, consideremos los sistemas *anti-spam* de los correos electrónicos. Durante su entrenamiento, a estos sistemas se les suministra una serie de mensajes ya clasificados, es decir, se les indica cuáles son *spam* y cuáles no. Una vez entrenado, el modelo es capaz de clasificar nuevos correos en función de lo aprendido, determinando si son *spam* o no.

Por otro lado, tenemos el *aprendizaje no supervisado*. A diferencia del supervisado, en este caso no se proporcionan etiquetas junto con los datos. El objetivo aquí es que la máquina descubra por sí misma estructuras y relaciones en el conjunto de datos. Una aplicación común de esta técnica se ve en el marketing de las empresas de retail. A partir de datos históricos de compras de sus clientes, los algoritmos pueden segmentar a los clientes en diferentes grupos basados en sus patrones de compra y, de esta manera, ofrecerles productos o servicios que se alineen con sus preferencias.

Finalmente, el *aprendizaje por refuerzo* se centra en permitir que una máquina o agente

aprenda por medio de la experiencia. A través de un sistema de recompensas y penalizaciones, la máquina es incentivada a tomar decisiones que maximicen alguna recompensa a lo largo del tiempo. Un ejemplo clásico de este tipo de aprendizaje se observa en juegos como el ajedrez. La inteligencia artificial detrás del rival virtual se entrena jugando miles de partidas, aprendiendo de cada movimiento, y ajustando su estrategia para mejorar su rendimiento en partidas futuras.

La taxonomía tradicional del aprendizaje automático, que se basa en cómo se presenta el problema a resolver (como supervisado, no supervisado y por refuerzo), no es la única manera de clasificar los algoritmos. En efecto, si nos enfocamos en las técnicas específicas que se utilizan para aprender, nos encontramos con una serie de categorías que arrojan luz sobre la rica variedad de enfoques en este campo.

Una de estas técnicas es el *Multi-Task Learning* [Crawshaw, 2020]. Enmarcada dentro del aprendizaje supervisado, esta técnica entrena un modelo para que, a partir de un único conjunto de datos, pueda aprender múltiples tareas relacionadas simultáneamente. Imaginemos un modelo entrenado con un conjunto de imágenes, que no solo identifica si en una foto hay un perro, sino también si es de día o de noche, y si hay algún otro animal, como un gato, presente.

Otra técnica es el *Active Learning* [Settles, 2009]. Esta estrategia involucra una colaboración activa entre el modelo y un operador humano para desambiguar y mejorar la calidad del aprendizaje. Es especialmente útil en escenarios donde los datos son limitados o donde adquirir nuevos datos es costoso. En tales situaciones, la máquina puede solicitar la intervención humana para esclarecer o confirmar ciertas predicciones o clasificaciones.

El *Online Learning* [Saad, 1999] es una técnica que permite a los modelos adaptarse a lo largo del tiempo. En lugar de entrenar un modelo desde cero cada vez que hay nuevos datos, el modelo se actualiza continuamente a medida que llega nueva información. Esta técnica es crucial en entornos donde la distribución de los datos cambia con el tiempo, como puede ser el volátil mercado financiero.

Transfer Learning [Bozinovski y Fulgosi, 1976] es una estrategia que aprovecha el conocimiento adquirido de un modelo previamente entrenado para una tarea y lo aplica en una nueva tarea relacionada. Por ejemplo, si tenemos un modelo entrenado para detectar enfermedades en general en pacientes, con *Transfer Learning*, este modelo podría ser adaptado para identificar enfermedades específicas sin tener que empezar el entrenamiento desde cero.

Finalmente, el *Ensemble Learning* [Maclin y Opitz, 2011] se refiere al uso de múltiples modelos para abordar una tarea particular. En lugar de confiar en un solo modelo, se entrenan varios, y sus predicciones se combinan, a menudo promediándolas o votando, para producir una salida final. Esta técnica puede mejorar la precisión y robustez de las predicciones.

La razón de mencionar ambas clasificaciones en el aprendizaje automático radica en el propó-

sito central de este trabajo: comprender, explicar y proponer una solución para el fenómeno del *olvido catastrófico*, particularmente evidente en el aprendizaje supervisado cuando se aplica bajo un enfoque de aprendizaje online denominado *aprendizaje continuo*.

El aprendizaje continuo se centra en enseñar a un modelo a dominar tareas de manera individual y secuencial. Es decir, el modelo se capacita primero en una tarea y, una vez completado este entrenamiento, avanza a la siguiente. Sin embargo, una característica inherente de este proceso es que al aprender una nueva tarea, el modelo tiende a perder u *olvidar* la información o habilidades adquiridas en las tareas anteriores. A este fenómeno se le conoce como **olvido catastrófico**.

Lo que es aún más fascinante es que este fenómeno se manifiesta con particular intensidad en estructuras de redes neuronales inspiradas en el funcionamiento del cerebro humano, específicamente en redes conexionistas. Estas redes, que imitan la estructura y dinámica de las neuronas biológicas, parecen, paradójicamente, ser especialmente susceptibles a este desafío del olvido al adquirir nuevos conocimientos.

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

La interferencia catastrófica, también conocida como olvido catastrófico es la tendencia de una red neuronal artificial a olvidar drásticamente información adquirida previamente a medida que nueva información se va presentando. Este fenómeno es inherente a la arquitectura que presenta una red neuronal en conjunto con el tipo de algoritmo utilizado para realizar correcciones en la inferencia del modelo cuando este se equivoca. Generalmente este algoritmo es la propagación hacia atrás, también conocido como *backpropagation*.

1.1. ¿Cómo se genera el fenómeno en estudio?

Una red neuronal artificial y cualquier derivado de este almacena la información necesaria para realizar inferencias en los parámetros o pesos internos. Durante el proceso de entrenamiento, estos pesos son actualizados para ajustarse mejor a los datos que le son proveídos. Para determinar qué pesos deben ser corregidos y a qué valor deben ser actualizados se utiliza el algoritmo de *backpropagation* que es capaz de determinar, a través de la minimización de una función de costo, por medio de la regla de la cadena [Cha, 2024a] (concepto matemático para explicar cómo se puede obtener la derivada de un valor que es dependencia de otros valores), con qué valor se debe actualizar los pesos de la red.

El problema de este algoritmo es que, de forma nativa, no establece un criterio para determinar si el cambio de valor de un parámetro de la red puede afectar negativamente en la inferencia de tareas previamente aprendidas. ¿Por qué? pues este algoritmo es libre de contexto, es decir, no necesita saber nada acerca de los pesos más que su dependencia en una función de costo que se busca minimizar.

El problema no es del algoritmo de *backpropagation* por si solo, sino que de la combinación de tres elementos del contexto presentado. Estos son:

1. Arquitectura del modelo basada en redes neuronales artificiales
2. Algoritmo de actualización de pesos *backpropagation*
3. Entrenamiento continuo de forma secuencial

La solución trivial en este contexto es cambiar el punto 3, es decir, en lugar de realizar aprendizaje continuo, realizar un entrenamiento *offline* o *batch learning* pero esto rompe el propósito del aprendizaje continuo.

1.2. ¿Por qué aprendizaje continuo?

Como se indicó al final de la subsección 1.1, cambiar el tipo de entrenamiento no es factible. No es factible por una variedad de razones las cuales serán abordadas a continuación

1.2.1. Disponibilidad de datos

Gran parte de la comunidad dedicada al aprendizaje de máquina, ciencia de datos y áreas relacionas se ha habituado a partir con un conjunto de datos, realizar un procesamiento de estos con fines de mejorar el desempeño del modelo, aplicar técnicas para los hiperparámetros que maximizan el desempeño del modelo respecto a alguna métrica establecida, entrenar el modelo y luego ver el desempeño del modelo frente a datos que jamas ha visto. Todo este proceso parte de la base de que el conjunto de datos está completamente disponible. La realidad es que no siempre se tiene todos los datos al comienzo del problema y sin embargo es necesario tener un modelo que por lo menos funcione con los datos actualmente disponibles.

La intuición indica que cuando lleguen nuevos datos, se podría reentrenar el modelo con los datos ya disponibles en conjunto con los nuevos y así se soluciona el problema de la disponibilidad de datos. Este es un pensamiento un tanto ingenuo que se explica en el siguiente punto.

1.2.2. Privacidad de los datos

Poseer el total de los datos, ya sea completamente desde el inicio u obtenidos parcialmente e ir almacenándolos hasta agruparlos todos, es el caso ideal. Seguramente el modelo que deseamos entrenar tendrá un resultado mejor que si entrenásemos de forma secuencial, sin embargo, para hacer esto se está asumiendo que se tiene el derecho de almacenamiento de los datos. No siempre se puede almacenar los datos, ya sea por leyes gubernamentales, políticas internacionales u cláusulas de un contrato lo que implica que a medida que se obtienen los datos, estos deben ser inmediatamente utilizados para entrenar al modelo y posteriormente desechados sin dejar rastros de estos en el sistema.

Se puede solventar este percance solicitando al organismo acreedor de los datos, si estos poseen los permisos necesarios, que almacenen los datos hasta que estén completos y luego dejarlos disponibles solo para realizar el entrenamiento. Sin embargo aún cuando fuésemos capaces de solventar estos problemas, podría generarse un conflicto con el siguiente punto.

1.2.3. Optimización de recursos

No existe nada más valioso para el mundo como el tiempo, pues este es un recurso que no se puede recuperar. En términos computacionales el tiempo también es un recurso valioso y en conjunto con este existen otros recursos computacionales de los que se debe ser cuidadoso por la relevancia que tienen. Cuando realizamos aprendizaje continuo, también se quiere ser lo más eficiente en el manejo de recursos utilizados, estos recursos son:

Tiempo

No se quiere tener que esperar un cantidad considerable de tiempo para obtener los primeros resultados. Esto aplica a esperar a recibir todos los datos al mismo tiempo pero también al tiempo computable para obtener resultados.

Memoria y procesamiento

Desde que se inventaron los computadores, han evolucionado significativamente en términos de capacidad de almacenamiento y procesamiento. Desde que ya no tenemos que lidiar con almacenar información eficientemente en un *diskette* de 1.44 MB o 720KB, o lograr que un videojuego pueda ejecutarse en una consola de mínimas capacidades, hemos aprovechado estos recursos para desarrollar tecnologías más avanzadas.

Hoy en día, existen unidades de persistencia con capacidad de almacenamiento de miles o hasta millones de veces el tamaño de *diskettes*. En lugar de enfocarnos en encontrar formas de almacenar más información en menos espacio, hemos se centrado en abordar otros problemas que han surgido con la tecnología avanzada.

En teoría, las memorias de acceso aleatorio son capaces de mantener información de muchos procesos o unidades de procesamiento lógicas que pueden correr en paralelo a un bajo costo. Esto significa que, aunque no siempre es necesario tener una alta disponibilidad de memoria y procesamiento, es importante considerar estos factores al diseñar sistemas y aplicaciones para garantizar su eficiencia y rendimiento.

Todos estos puntos son parte de las causas del fenómeno de olvido catastrófico, el olvido catastrófico el problema central, y las consecuencias son el no querer utilizar aprendizaje continuo a pesar de los beneficios, en términos de recursos computacionales, que este trae. Una representación de lo anteriormente descrito se puede ver en la Figura 1

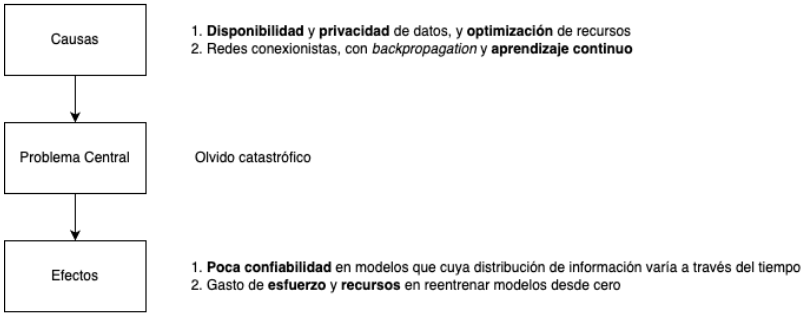


Figura 1: Árbol del problema

1.3. Objetivos

1.3.1. Objetivo general

Diseñar y evaluar estrategias de aprendizaje de máquina que reduzcan el fenómeno de olvido catastrófico en un contexto de aprendizaje continuo en clasificación de imágenes

1.3.2. Objetivo específico

1. Evaluar distintas técnicas del estado del arte para resolver el olvido catastrófico, midiendo diferentes métricas relacionadas al aprendizaje continuo
2. Proponer y evaluar estrategias para enfrentar el olvido catastrófico en tareas de clasificación.
3. Minimizar el uso de recursos computacionales en el entrenamiento e inferencia con las estrategias propuestas.

1.4. Alcance

El alcance del presente trabajo abarca el aprendizaje de tareas de clasificación de imágenes, en un contexto de aprendizaje supervisado de forma secuencial, es decir, dentro de un contexto de aprendizaje continuo. Se restringe más el área a aprendizaje continuo con *class-incremental learning*, concepto abordado más adelante. En esta investigación no considera la parte que comprende el entendimiento y procesamiento de los datos, ni uso del modelo en entornos de producción, es decir, solamente fines de exploración e investigación. ¿Por qué? a pesar de que se consideran elementos claves para obtener éxito en la resolución de un problema utilizando algoritmos de aprendizaje automático, estos añaden una complejidad innecesaria al estudio, complejidad que no es foco principal ni secundario, de este

modo no entra preprocesamiento de los datos de entrada, análisis de los datos de entrada para realizar *feature engineering* o *feature selection*.

Dado que el aprendizaje continuo no es una técnica inherente del aprendizaje supervisado, el abanico de técnicas aumenta y no es prioridad en este trabajo dar foco a estas técnicas, por lo que no entran dentro del alcance el aprendizaje semisupervisado, aprendizaje no supervisado o aprendizaje reforzado.

El hecho de que no entren conceptos como foco de estudio no significa que no se realicen en algún punto de forma explícita o implícita. En esta investigación, el conjunto de datos de entrenamiento y prueba pasa por una transformación, específicamente estandarización.

CAPÍTULO 2
MARCO CONCEPTUAL

2.1. Definiciones

2.1.1. Modelo de Aprendizaje de máquinas

Un modelo de aprendizaje de máquinas o *machine learning* es un algoritmo o una representación matemática que, después de un proceso de entrenamiento basado en datos, es capaz de realizar predicciones o decisiones sin recibir instrucciones externas para ello. Estos modelos encuentran y aprenden los patrones y estructuras a partir de los datos de entrenamiento para luego hacer inferencias sobre datos nuevos. En esencia, un modelo de *machine learning* emula la capacidad humana de aprender a partir de la experiencia y usar ese aprendizaje para realizar tareas específicas. Otra definición más a fin con el presente trabajo es la siguiente:

> Se dice que un programa de computador aprende de una experiencia E con respecto a una clase de tareas T y medida de desempeño P , si su desempeño en las tareas T , medido mediante P , mejora con la experiencia E [Mitchell, 1997].

Notar que esta definición es agnóstica al tipo de aprendizaje, es decir, aprendizaje supervisado, no supervisado y reforzados calzan con esta definición.

2.1.2. Entrenamiento

El entrenamiento, en el contexto de *machine learning*, se refiere al proceso mediante el cual un modelo algorítmico ajusta y optimiza sus parámetros internos utilizando un conjunto de datos conocido como conjunto de entrenamiento. Durante este proceso, el modelo *aprende* al intentar minimizar las diferencias entre sus predicciones y los valores reales presentes en el conjunto de datos, adaptando gradualmente sus parámetros con el objetivo de mejorar su precisión. El resultado del entrenamiento es un modelo que ha internalizado patrones y relaciones de los datos, permitiéndole hacer predicciones o clasificaciones sobre nuevos datos no vistos anteriormente.

En otros términos, el entrenamiento es el proceso en donde una máquina de aprendizaje M aprende una tarea T mediante la experiencia E.

2.1.3. Parámetros

En el contexto de *machine learning*, los parámetros son variables internas del modelo que se ajustan durante el proceso de entrenamiento con el objetivo de mejorar la predicción o clasificación del modelo. Estos parámetros determinan la configuración específica del modelo y su comportamiento al recibir nuevos datos. Por ejemplo, en una regresión lineal, los parámetros son los coeficientes y el término independiente que define la línea. En una red neuronal, los parámetros incluyen los pesos y los sesgos de las neuronas. A medida que el modelo se entrena, estos parámetros se actualizan y optimizan para minimizar el error entre las predicciones del modelo y los valores reales del conjunto de entrenamiento.

2.1.4. Función de pérdida

La función de pérdida (*loss function*), también conocida como función de costo o función de error, es una función que mide la discrepancia entre la predicción realizada por un modelo de *machine learning* y el valor real o verdadero. Se utiliza durante el proceso de entrenamiento para guiar la optimización de los parámetros del modelo. Específicamente, el objetivo del entrenamiento es minimizar el valor de esta función.

La elección de la función de pérdida depende de la naturaleza del problema. Por ejemplo:

En problemas de regresión, comúnmente se utiliza la función *Mean Square Error (MSE)*. Esta función mide el promedio de la diferencia al cuadrado entre un valor predicho y el valor real, tal como se expresa en la ecuación (1)

$$\mathsf{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{1}$$

En problemas de clasificación binaria, una función típicamente usada es *Hinge Loss*, que mide el margen entre la etiqueta real y la etiqueta predicha. Su definición se puede ver en la ecuación (2).

$$l(y, \hat{y}) = \mathsf{max}(0, 1 - \hat{y} \cdot y) \tag{2}$$

En problemas de clasificación multiclase, una función comúnmente utilizada es *Cross Entropy*, esta mide la diferencia entre la probabilidad predicha para cada clase y la probabilidad real de cada clase. Su definición se encuentra en la ecuación (3).

$$\mathsf{CE}(y, \hat{y}) = -\sum_{c=1}^{C} y_c \cdot \mathsf{log}(\hat{y}_c) \tag{3}$$

Al calcular el valor de la función de pérdida para un conjunto de datos y luego tomar medidas para minimizarlo (a través de técnicas de optimización como el descenso de gradiente), se mejora el rendimiento del modelo, permitiendo que sus predicciones sean más precisas.

2.1.5. Regularizadores

Un regularizador es una técnica utilizada en el entrenamiento de modelos de machine learning y estadísticas para prevenir el sobreajuste (*overfitting*), es decir, la adaptación excesiva del modelo a los datos de entrenamiento, lo que puede hacer que el modelo tenga un rendimiento deficiente en datos no vistos. Al añadir un término de penalización a la función de pérdida, el regularizador restringe la capacidad del modelo, favoreciendo soluciones más simples o restringiendo la magnitud de los parámetros del modelo.

Existen diferentes tipos de regularizaciones, y aquí se mencionan algunas de las más comunes:

1. **Regularización L1 (Lasso):** Añade una penalización equivalente al valor absoluto de la magnitud de los coeficientes. Esto puede llevar a que algunos coeficientes se vuelvan exactamente cero, resultando en un modelo más disperso.
2. **Regularización L2 (Ridge):** Añade una penalización equivalente al cuadrado de la magnitud de los coeficientes. Esto tiende a reducir la magnitud de los coeficientes pero no necesariamente los hace cero.
3. **Regularización Elastic Net:** Combina las penalizaciones L1 y L2. Es útil cuando hay múltiples características que están correlacionadas entre sí.
4. **Regularización de dropout:** Utilizado principalmente en redes neuronales, donde durante el entrenamiento se *apagan* aleatoriamente ciertas neuronas, lo que significa que durante una determinada iteración de entrenamiento, estas neuronas no inciden en la predicción y tampoco se actualizan. Esto evita que la red se vuelva demasiado dependiente de cualquier neurona individual y favorece una distribución más uniforme del aprendizaje.

La regularización introduce un equilibrio entre ajustar los datos lo mejor posible y mantener el modelo simple para garantizar una buena generalización a nuevos datos. El grado de regularización suele ser controlado por un hiperparámetro, que se ajusta según el rendimiento del modelo en un conjunto de validación.

2.1.6. Optimizadores

Un optimizador es una técnica utilizada en el proceso de aprendizaje automático para encontrar los parámetros del modelo que minimizan o maximizan una función objetivo. En

otras palabras, el objetivo del optimizador es encontrar los valores más adecuados para los parámetros del modelo para mejorar la precisión de las predicciones o para maximizar la rentabilidad de un sistema.

Existen diferentes tipos de optimizadores, algunos de los más comunes son:

1. *Gradient Descent (GD)*: Es un optimizador de línea basado en el método del gradiente. *GD* encuentra los valores para los parámetros que minimizan la función objetivo, que se define como una función de pérdida o error entre las predicciones y los datos de entrada [Chong y Żak, 2013].

2. *Stochastic Gradient Descent (SGD)*: Es un optimizador similar a *GD*, pero en lugar de utilizar todos los datos de entrenamiento para calcular el gradiente, se utiliza una muestra aleatoria de ellos. Esto mejora la eficiencia del algoritmo y reduce la cantidad de memoria requerida [Amari, 1993].

3. *Adam*: Es un optimizador adaptativo que combina elementos de *GD* y *SGD*. *Adam* tiene una constante de aprendizaje que se ajusta automáticamente en función del tamaño del conjunto de datos y del tamaño del modelo [Kingma y Ba, 2014].

4. *RMSProp*: Es un optimizador similar a *Adam*, pero con una forma única de calcular la tasa de aprendizaje. *RMSProp* utiliza la raíz cuadrada media de los gradientes históricos para calcular la tasa de aprendizaje [Bushaev, 2018].

5. *Momentum*: Es un optimizador que combina elementos de *GD* y *SLGD*. *Momentum* utiliza la velocidad actual del modelo para calcular el gradiente, en lugar de utilizar solo los datos de entrenamiento más recientes [mom, 2021].

6. *Nesterov Accelerated Gradient (NAG)*: Es un optimizador similar a *GD*, pero con una forma única de calcular la tasa de aprendizaje que incluye un término de *aceleración* que mejora la eficiencia del algoritmo [nag, 2021].

7. *Adagrad*: Es un optimizador que utiliza una función no lineal para calcular el gradiente, en lugar de utilizar una función lineal como *GD*. *Adagrad* utiliza una función de pérdida no lineal que se ajusta automáticamente en función del tamaño del modelo y del conjunto de datos [Duchi *et al.*, 2011].

8. *Adadelta*: Es un optimizador similar a *Adagrad*, pero con una forma única de calcular la tasa de aprendizaje. *Adadelta* utiliza una función no lineal para calcular el gradiente y una constante de aprendizaje que se ajusta automáticamente en función del tamaño del modelo y del conjunto de datos [Zeiler, 2012].

Cada optimizador tiene sus ventajas y desventajas, y la elección del optimizador adecuado depende del problema específico que se está intentando resolver y de las características del modelo y del conjunto de datos de entrenamiento.

2.1.7. Hiperparámetros

Los hiperparámetros son aquellos parámetros que no se aprenden directamente dentro de los estimadores en el proceso de entrenamiento de modelos de machine learning. En cambio, son valores definidos previamente o ajustados externamente antes de comenzar el proceso de entrenamiento. Su propósito es ayudar a controlar y guiar el proceso de entrenamiento.

Los hiperparámetros difieren de los parámetros regulares (como los pesos y sesgos en las redes neuronales) en que estos últimos son aprendidos directamente de los datos durante el entrenamiento, mientras que los hiperparámetros son configurados para influenciar la forma en que el modelo se entrena.

Algunos ejemplos comunes de hiperparámetros incluyen:

1. **Tasa de aprendizaje:** Es un multiplicador que determina el tamaño del paso que se toma en cada actualización de los parámetros durante el entrenamiento. Una tasa de aprendizaje alta puede hacer que el entrenamiento converja rápidamente, pero también puede provocar oscilaciones o divergencia. Una tasa baja puede llevar a una convergencia más estable, pero puede ser demasiado lenta.
2. **Número de épocas:** Es el número de veces que el algoritmo de aprendizaje trabajará en todo el conjunto de entrenamiento.
3. **Tamaño del lote (batch size):** Es el número de muestras de datos utilizadas en una actualización del modelo para el entrenamiento estocástico.
4. **Momentum:** Utilizado en optimizadores como *SGD* con *momentum*, ayuda a acelerar la convergencia y superar mínimos locales.
5. **Regularización:** Parámetros como el coeficiente de regularización L1 o L2 que ayudan a prevenir el sobreajuste penalizando ciertos patrones de parámetros.
6. **Arquitectura del modelo:** En el caso de redes neuronales, esto puede incluir el número de capas ocultas, el número de neuronas por capa, tipos de activaciones, etc.
7. **Constantes del modelo**: El modelo puede definir constantes usadas para su aprendizaje que no forman parte de los elementos anteriormente enumerados. En el caso de las estrategias que se verán en este trabajo pueden ser valores λ que establecer el grado de *apego* a una solución anterior.

Ajustar hiperparámetros correctamente es crucial ya que pueden tener un gran impacto en la calidad del modelo entrenado. El proceso de seleccionar los valores óptimos para estos hiperparámetros se conoce como *ajuste de hiperparámetros* o *tuning*, y puede realizarse mediante técnicas como la búsqueda en malla (grid search), búsqueda aleatoria (random search) o métodos más avanzados como optimización bayesiana.

2.1.8. Aprendizaje Continuo

El aprendizaje continuo, también conocido como aprendizaje incremental o aprendizaje a lo largo de la vida (*lifelong learning* en inglés), se refiere a la capacidad de un modelo de *machine learning* de aprender de manera continua a partir de datos nuevos, acumulativos o secuenciales, sin olvidar las representaciones o patrones aprendidos previamente. Es un paradigma que intenta replicar la forma en que los seres humanos aprenden y adaptan su conocimiento a lo largo del tiempo, integrando nueva información a lo ya conocido sin olvidar el conocimiento adquirido en una etapa previa.

2.1.9. Taxonomia en Aprendizaje Continuo

2.1.9.1 Task-Incremental Learning

El aprendizaje *task-incremental* se enfoca en mejorar el rendimiento de una máquina de aprendizaje en un conjunto de tareas específicas, sin afectar su capacidad para realizar otras tareas. El objetivo es ajustar la configuración de la máquina para mejorar su desempeño en una serie de tareas relacionadas, pero no necesariamente similares.

Por ejemplo, si se tiene una máquina de aprendizaje que puede clasificar imágenes de perros y gatos, el aprendizaje *task-incremental* se enfocaría en mejorar su capacidad para clasificar imágenes de nuevos tipos de animales, como conejos o avestruces.

2.1.9.2 Class-Incremental Learning

El aprendizaje *class-incremental* se enfoca en mejorar la capacidad de una máquina de aprendizaje para clasificar objetos pertenecientes a nuevos conjuntos de clases, sin afectar su rendimiento en clases previas. El objetivo es ajustar la configuración de la máquina para mejorar su desempeño en una serie de clases relacionadas, pero no necesariamente similares.

Por ejemplo, si se tiene una máquina de aprendizaje que puede clasificar imágenes de personas, el aprendizaje *class-incremental* se enfocaría en mejorar su capacidad para clasificar imágenes de nuevos tipos de personas, como niños o ancianos.

2.1.9.3 Domain-Incremental Learning

El aprendizaje *domain-incremental* se enfoca en mejorar la capacidad de una máquina de aprendizaje para realizar tareas relacionadas pero no exactamente el mismo tipo de tareas. En este tipo de escenarios, la estructura del problema a enfrentar es estático pero el dominio

cambia. ¿A qué nos referimos con estructura? la forma de la entrada de los datos y la forma de la salida, es decir, todas las tareas contienen datos con la misma dimensionalidad, sin embargo, el contenido es el que cambia.

Por ejemplo, si se tiene una máquina de aprendizaje que puede dentro de una tarea puede discriminar si una imagen contiene o un perro o un gato donde el resultado 0 implica que es gato y 1 perro, en una siguiente tarea su labor puede ser discriminar si una imagen contiene un un lobo o un león, donde 0 es león y 1 lobo. Notar que durante una evaluación, la máquina no sabe a qué tarea pertenece una imagen pero si debe ser capaz de responder 0 o 1 correctamente. Al hecho de no saber a que tarea corresponde una imagen, y de hecho ignorar ese dato se le conoce como *task-agnostic learning*.

Un ejemplo gráfica de la diferencia de estas categorias en el contexto de este trabajo se aprecia en la Figura 2. La explicación es la siguiente:

- **Task Incremental**: Se tiene un modelo ya entrenado para clasificar imágenes como 0 o 1, donde la salida del modelo es 0 y 1 para cada clase respectivamente. Posteriormente llega una nueva tarea que exige al modelo clasificar imágenes como 2 o 3, cuya salida también es 0 o 1. En el momento del entrenamiento se le indica al modelo a que tarea corresponde la imagen que está clasificando. Cuando se usa el modelo para realizar inferencias, también se suele indicar a qué tarea corresponde la imagen a inferir.
- **Domain Incremental**: Se tiene un modelo ya entrenado para clasificar imágenes como 0 o 1, donde la salida del modelo es 0 y 1 para cada clase respectivamente. Posteriormente llega una nueva tarea que exige al modelo clasificar imágenes como 2 o 3, cuya salida también es 0 o 1, es decir, la misma descripción que *task incremental*, sin embargo, el modelo no se le indica si la imagen pertenece a una tarea u otra, porque es irrelevante. El propósito de este enfoque es que el modelo, sin ser consciente de la tarea, realice una inferencia sobre una imagen como 0 o 1, luego el usuario final es quien debe entender que esa salida corresponde a una tarea en específica.
- **Class Incremental**: Se tiene un modelo ya entrenado para clasificar imágenes como 0 o 1, donde la salida del modelo es 0 y 1 para cada clase respectivamente. Posteriormente llega una nueva tarea que exige al modelo clasificar imágenes como 2 o 3, pero ahora la salida es 2 o 3, es decir, ya no reutiliza la misma salida para tareas diferentes.

En resumen, mientras que el aprendizaje *task-incremental* y *class-incremental* se enfocan en mejorar la capacidad de una máquina de aprendizaje para realizar tareas específicas, el aprendizaje *domain-incremental* busca mejorar su capacidad para adaptarse a nuevas tareas sin importar su tipo.

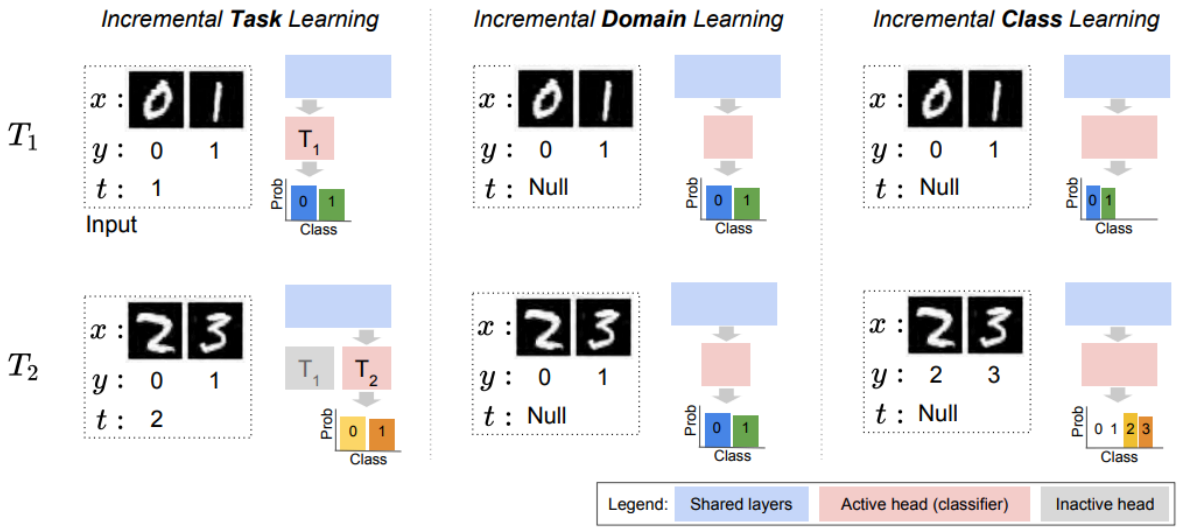


Figura 2: Diferencia entre Task, Class y Domain incremental Learning
Elaboración extraída de [Hsu *et al.*, 2018]

2.1.10. Redes Neuronales / Conexionistas

Las redes conexionistas, también conocidas como redes neuronales artificiales (ANN por sus siglas en inglés, de *Artificial Neural Networks*), son modelos computacionales inspirados en la estructura y funcionamiento de las redes neuronales biológicas presentes en el cerebro. Estos modelos se componen de unidades o nodos, llamados *neuronas*, que están interconectadas a través de conexiones ponderadas.

Características clave de las redes conexionistas:

1. **Unidades de Procesamiento:** Son los nodos o neuronas individuales que componen la red. Cada unidad toma una serie de entradas, las procesa y produce una salida. Esta salida es una función, generalmente no lineal, de la suma ponderada de sus entradas.
2. **Conexiones:** Las unidades en una red están conectadas entre sí mediante conexiones que llevan asociado un peso. Estos pesos son ajustables y representan la *fuerza* o *influencia* de una conexión. El proceso de aprendizaje en una red conexionista implica ajustar estos pesos para mejorar el desempeño de la red en una tarea determinada.
3. **Aprendizaje:** Las redes conexionistas aprenden a partir de ejemplos. Durante el proceso de entrenamiento, se ajustan los pesos de las conexiones para minimizar el error entre las salidas producidas por la red y las salidas deseadas.
4. **Capacidad de generalización:** Una vez entrenadas, las redes neuronales pueden generalizar su aprendizaje a ejemplos no vistos previamente, lo que las hace poderosas para tareas como la clasificación, predicción y reconocimiento de patrones.

5. **Tolerancia a fallos:** Debido a su naturaleza distribuida, las redes conexionistas pueden ser robustas frente a fallos parciales, es decir, la eliminación o fallo de algunas neuronas no suele deshabilitar por completo la funcionalidad de la red.

El término *conexionista* proviene del hecho de que el conocimiento en estas redes está almacenado en las conexiones (pesos) entre las unidades, en lugar de estar almacenado en las unidades mismas. Estas redes son la base de muchos avances recientes en inteligencia artificial, especialmente en áreas como el aprendizaje profundo (*deep learning*).

2.1.11. Olvido Catastrófico

El *olvido catastrófico* se refiere a un problema observado en redes neuronales y otros modelos de aprendizaje automático cuando se les entrena secuencialmente en varias tareas. Específicamente, al entrenar una red en una nueva tarea, puede *olvidar* rápidamente la información relacionada con tareas anteriores. Esto significa que su rendimiento en tareas previamente aprendidas se deteriora significativamente a medida que aprende nuevas tareas.

Algunas características del olvido catastrófico incluyen:

1. **Rápida degradación del rendimiento:** Una vez que se comienza el entrenamiento en una nueva tarea, el desempeño en tareas anteriores puede deteriorarse rápidamente.
2. **Superposición de representaciones:** En redes neuronales, especialmente en aquellas con arquitecturas de aprendizaje profundo, las representaciones internas de diferentes tareas pueden superponerse. Cuando se ajustan los pesos para una nueva tarea, se pueden modificar inadvertidamente las representaciones útiles para tareas anteriores, llevando al olvido.
3. **Desafío en aprendizaje continuo:** El olvido catastrófico es un obstáculo importante en el campo del aprendizaje continuo, donde se espera que los modelos aprendan de forma continua y secuencial a lo largo del tiempo sin olvidar las tareas previas.

2.1.12. Propagación hacia adelante

Forward propagation (o propagación hacia adelante) se refiere al proceso mediante el cual una red neuronal procesa una entrada y avanza a través de las capas de la red para producir una salida. Es el mecanismo básico por el cual las redes neuronales realizan sus predicciones o inferencias. *Forward propagation* se lleva a cabo en cada iteración del entrenamiento y también cuando el modelo ya entrenado realiza predicciones en datos nuevos.

Para entender *forward propagation*, considere los siguientes pasos básicos:

1. **Entrada de Datos:** Se inicia introduciendo un vector de entrada en la capa de entrada de la red neuronal.

2. **Aplicación de Pesos y *Biases*:** Cada conexión entre neuronas tiene un peso asociado. Estos pesos se multiplican por las entradas para determinar la influencia de una neurona en la siguiente. A este producto se le añade un valor conocido como *bias* (sesgo).

3. **Función de Activación:** El resultado de la multiplicación de los pesos por las entradas y la adición del *bias* se pasa por una función de activación. Esta función introduce no-linealidades que permiten a la red neuronal modelar relaciones complejas. Algunas funciones de activación populares son la función sigmoide, *ReLU* (*Rectified Linear Unit*) y *tanh* (Tangente Hiperbólica).

4. **Repetir para Todas las Capas:** El proceso se repite para cada capa intermedia (oculta) de la red hasta que se alcanza la capa de salida.

5. **Producción de la Salida:** Una vez que la señal ha sido procesada a través de todas las capas de la red, se obtiene una salida en la capa final. Esta salida puede ser una predicción, una clasificación o cualquier otro tipo de resultado, según la tarea que la red esté diseñada para realizar.

Durante el proceso de entrenamiento, después de *forward propagation*, se calcula el error del modelo (usando una función de pérdida) comparando la salida obtenida con la salida esperada. Posteriormente, se utiliza este error para ajustar los pesos y *biases* de la red en un proceso conocido como *backpropagation* o propagación hacia atrás.

2.1.13. Propagación hacia atrás

Backward propagation (o *backpropagation*, propagación hacia atrás) es un algoritmo utilizado para entrenar redes neuronales artificiales. Es un método supervisado que emplea la regla de la cadena del cálculo diferencial para actualizar los pesos y sesgos de la red. La idea central de *backpropagation* es minimizar el error generado en las predicciones del modelo mediante la corrección sistemática de sus pesos y sesgos, propagando el error desde la salida de la red hacia sus entradas.

Los pasos básicos del proceso de *backpropagation* son los siguientes:

1. ***Forward Propagation*:** Se realiza una propagación hacia adelante de la entrada a través de la red para producir una predicción.

2. **Cálculo del Error:** Se calcula el error o la pérdida utilizando una función de pérdida, comparando la predicción obtenida con la verdad fundamental o el valor objetivo.

3. **Propagación del Error hacia Atrás:** A partir de la capa de salida y moviéndose hacia atrás a través de la red, se calculan las derivadas parciales del error con respecto a cada peso utilizando la regla de la cadena. Esto esencialmente determina cuánto contribuyó cada peso al error total.

4. **Actualización de Pesos y *Biases*:** Con las derivadas parciales calculadas y utilizando un algoritmo de optimización (como el descenso del gradiente), se actualizan los pesos y *biases* de la red para reducir el error en la siguiente iteración.

5. **Iteración:** Se repiten los pasos anteriores para múltiples entradas y durante varias épocas (recorridos completos del conjunto de datos) hasta que el error de la red alcance un valor satisfactoriamente bajo o hasta que no se observen mejoras significativas.

Una representación a través de pseucódigo para un proceso de entrenamiento completo utilizando *backpropagation* se puede apreciar en el algoritmo 1

Algorithm 1 Backpropagation

Require: Training set $\mathcal{D}$, learning rate η
Ensure: Trained neural network parameters W, b
Initialize weights W and biases b randomly
for each epoch **do**
  for each training sample $(x, y) \in \mathcal{D}$ **do**
    Forward Propagation:
    $a^{(1)} = x$
    for $l = 2, 3, \ldots, L$ **do**
      $z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}$
      $a^{(l)} = \sigma(z^{(l)})$
    end for
    $\hat{y} = a^{(L)}$
    Backpropagation:
    $\delta^{(L)} = \nabla_{a^{(L)}} \mathcal{L}(\hat{y}, y) \odot \sigma'(z^{(L)})$
    for $l = L-1, L-2, \ldots, 2$ **do**
      $\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)} \odot \sigma'(z^{(l)})$
    end for
    Update Parameters:
    for $l = 1, 2, \ldots, L$ **do**
      $W^{(l)} \leftarrow W^{(l)} - \eta \delta^{(l)} (a^{(l-1)})^T$
      $b^{(l)} \leftarrow b^{(l)} - \eta \delta^{(l)}$
    end for
  end for
end for

Backpropagation es el pilar del aprendizaje en redes neuronales profundas y ha sido fundamental para el renacimiento y el éxito de las redes neuronales en una variedad de aplicacio-

nes en las últimas décadas. Es esencial entender *backpropagation* para comprender cómo se ajustan y optimizan las redes neuronales durante el entrenamiento.

2.2. Métricas

A la hora de escoger una o varias métricas para realizar análisis o comparación de modelos realizando aprendizaje continuo, no es adecuado hacer uso de métricas como el *accuracy*, *precision*, *recall* o *f1-score* a secas, ya que este tipo de métricas, si bien dan información muy valiosa cuando se realiza un entrenamiento de forma concurrente, no ayuda a entender como es que la red va olvidando o como las tareas aprendidas se van influenciando entre sí para retener y/o olvidar información previa o futura. Por eso, las métricas que se definen a continuación son más adecuadas para este contexto. ¿Por qué?, pues en el aprendizaje continuo no siempre es de total relevancia medir las métricas mencionadas previamente porque también nos interesa medir qué tanto ha olvidado un modelo o que tanto estará dispuesto a olvidar o como ha variado el *accuracy* a medida que se aprenden nuevas tareas.

El desempeño general se puede evaluar con un promedio de la exactitud (average accuracy AA) y promedio incremental de la exactitud (average incremental accuracy AIA).

Accuracy: Sea TP las predicciones hechas por un modelo M de forma correcta hacia una clase j y FP las predicciones de forma incorrecta, el *accuracy* de la clase j se define como:

$$a_j = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

De esta forma misma forma, $a_{k,j}$ representa el *accuracy* sobre una clase j después de haber entrenado la tarea k, donde $j \leq k$

Average Accuracy (AA):

$$AA_k = \frac{1}{k} \cdot \sum_{j=1}^{k} a_{k,j}$$

Esta métrica resuelve la pregunta: **¿Qué tan exacto es mi modelo?**

Average Incremental Accuracy (AIA):

$$AIA_k = \frac{1}{k} \cdot \sum_{i=1}^{k} AA_i$$

Esta métrica resuelve la pregunta: **¿Qué tan exacto es mi modelo?**

La estabilidad de la memoria se puede evaluar usando forgetting measure (FM) y backward transfer (BWT). El olvido de una tarea se obtiene calculando el máximo desempeño obtenido en el pasado y su desempeño actual.

$$f_{j,k} = \max_{i \in \{1,\dots k-1\}}(a_{i,j} - a_{k,j}), \forall j \leq k$$

Backward Transfer (BWT): El backward Transfer mide la influencia que implica aprender la k-ésima tarea sobre todas las tareas previamente aprendidas

$$BWT_k = \frac{1}{k-1}\sum_{j=1}^{k-1}(a_{k,j} - a_{j,j})$$

Esta métrica resuelve la pregunta: **¿Qué tanto mejora mi aprendizaje sobre las experiencias anteriores el aprender la experiencia actual?**

Forgetting Measure (FM): El Forgetting Measure mide la diferencia entre el máximo conocimiento obtenido en cada una de las tareas durante el proceso de entrenamiento y el conocimiento que tiene actualmente. Se define el olvido de la j-ésima tarea, luego de ser entrenado hasta la tarea k y el olvido medio luego de que la tarea k ya hubiese sido aprendida.

$$FM_k = \frac{1}{k-1}\sum_{j=1}^{k-1} f_{j,k}$$

La plasticidad del aprendizaje se puede evaluar por intransiense measure (IM) y forward transfer (FWT).

Forward Transfer (FWT): FWT evalua la influencia promedio de todas las viejas tareas en la actual k-ésima tarea

$$FWT_k = \frac{1}{k-1}\sum_{j=2}^{k}(a_{j,j} - \tilde{a}_j)$$

donde $\tilde{a}_j$ se define como el *accuracy* de clasificación de un modelo referencia entrenado con el conjunto $\mathcal{D}_j$ para la j-ésima tarea.

Esta métrica resuelve la pregunta: **¿Qué tanto mejora mi aprendizaje sobre las experiencias posteriores el aprender la experiencia actual?**

Intransiense Measure (IM): IM es definido como la inhabilidad de un modelo para aprender una nueva tarea, calculado como la diferencia de una tarea entre el desempeño de su entrenamiento articulado y su desempeño de aprendizaje continuo

$$IM_k = a_k^* - a_{k,k}$$

donde a_k^* es el *accuracy* en la clasificación con un modelo de referencia inicializado entrenado con $\cup_{j=1}^{k} \mathcal{D}_|$ para la k-ésima tarea.

Constantemente se proponen nuevas métricas para medir otras dimensiones en *continual learning* [Díaz-Rodríguez *et al.*, 2018], sin embargo, ninguna de ellas se utilizará en este trabajo.

2.3. Historia

El término *interferencia catastrófica* u *olvido catastrófico* se refiere a un fenómeno observado en redes conexionistas. Esta nomenclatura fue introducida por McCloskey y Cohen en 1989 [McCloskey y Cohen, 1989] durante su investigación dedicada a este particular fenómeno. En su estudio, se centraron en el entrenamiento secuencial de redes neuronales utilizando diferentes conjuntos de datos, revelando cómo el aprendizaje de nuevos datos puede comprometer o ïnterferirçon el conocimiento previamente adquirido.

En el primer experimento, los investigadores utilizaron una sencilla red neuronal y seleccionaron un conjunto de datos, denotado como S_1. Este conjunto comprendía 17 cuestiones básicas de aritmética con números de una sola cifra, tales como $1 + 2$, $1 + 3$, ..., $1 + 9$, y también $2 + 1$, $3 + 1$, ..., $9 + 1$. Como se podría anticipar, al entrenar la red con S_1, el error fue disminuyendo progresivamente, llegando al punto donde el modelo podía representar y resolver adecuadamente los problemas del conjunto. Posteriormente, se introdujo un segundo conjunto, S_2, que constaba de 17 nuevos problemas aritméticos basados en el número 2: $2+1$, $2+2$, ..., $2+9$, así como $1+2$, $2+2$, ..., $2+9$. Tras el entrenamiento con S_2, al evaluar la capacidad del modelo para representar y resolver estos nuevos problemas, no se encontraron dificultades. Sin embargo, al volver a testear el modelo con problemas del conjunto inicial, como $1 + 7$, este ya no podía proporcionar respuestas correctas. De hecho, las respuestas generadas se asemejaban más a números erróneos que a las soluciones correctas. Es relevante destacar que ejemplos como $1+2$ y $2+1$, que aparecían en ambos conjuntos S_1 y S_2, también resultaban problemáticos en las primeras etapas del entrenamiento basado en S_2.

En un segundo experimento orientado al ámbito conexionista, McCloskey y Cohen buscaron emular un estudio sobre interferencia retroactiva en seres humanos, originalmente llevado a cabo por Barnes y Underwood en 1959. Para ello, entrenaron un modelo utilizando dos conjuntos de listas, etiquetados como $\overline{AB}$ y $\overline{AC}$, acompañados de patrones de contexto en el vector de entrada. Estos patrones de contexto servían para diferenciar claramente entre las dos listas durante el proceso de entrenamiento. El objetivo era que, frente al estímulo A en el contexto AB, el modelo produjera la respuesta B, y, del mismo modo, al presentarle el estímulo A en el contexto AC, generara la respuesta C.

Al entrenar el modelo de manera concurrente (es decir, introduciendo ambas listas $\overline{AB}$ y $\overline{AC}$ al mismo tiempo), este demostró ser capaz de aprender y asociar adecuadamente los

estímulos y respuestas. Sin embargo, cuando se procedió con un entrenamiento secuencial (primero con $\overline{AB}$ y posteriormente con $\overline{AC}$), se observaron comportamientos distintos. Tras cada ciclo de entrenamiento con la lista $\overline{AC}$, se evaluó el rendimiento del modelo en relación con ambas listas.

Los resultados revelaron que el número de ciclos de entrenamiento sobre $\overline{AC}$ requeridos para alcanzar una tasa de respuestas correctas del 50 % en el modelo estudiado por Barnes y Underwood contrastaba marcadamente con el modelo basado en backpropagation, el cual mostró alrededor de un 0 % de respuestas correctas. Aun más, se observó que este último modelo, al ser expuesto a un dato para el que debería haber respondido con el patrón B, tendía a generar respuestas más cercanas al patrón C.

McCloskey y Cohen trataron de reducir la interferencia presentaba aplicando una serie de técnicas como cambiar el número de unidades escondidas, el hiperparámetro learning rate, sobre entrenar sobre la lista $\overline{AB}$, entre otras, pero ninguna pudo dar frente al fenómeno evidenciado.

Las conclusiones de los autores fueron que:

- Ocurrirá algo de interferencia cuando nuevo aprendizaje afecte al peso de las conexiones entre neuronas que simbolizan una representación del problema.
- A mayor cantidad de nueva información aprendida, mayor la disrrupción en el conocimiento previamente adquirido por la red.
- La interferencia fue catastrófica en redes con backpropagation cuando el entrenamiento fue secuencial en lugar de concurrente

Al año siguiente, Ratcliff investigó utilizando varios modelos que implementaban Backpropagation [Ratcliff, 1990]. Estos modelos se basaban en procedimientos estándar de memorias de reconocimiento, y en ellos, los ítems eran aprendidos de forma secuencial. Tras examinar el rendimiento de estos modelos de reconocimiento, Ratcliff identificó dos problemas fundamentales:

1. La información que había sido aprendida con eficacia era rápidamente olvidada a medida que se introducía y aprendía nueva información en la red. Este comportamiento se observó tanto en redes de pequeño como de gran tamaño. Sorprendentemente, incluso una única iteración de entrenamiento con nuevos datos resultaba en una significativa pérdida de información previamente adquirida. Además, Ratcliff observó que los resultados del modelo se convirtieron en una amalgama de la información nueva y la previamente aprendida. En modelos de mayor tamaño, los ítems que se aprendían en grupos (por ejemplo, $\overline{AB}, \overline{BC}, \overline{CD}, ...$) mostraban una mayor resistencia al olvido en comparación con aquellos ítems que se aprendían de manera individual (como $\overline{A}, \overline{B}, \overline{C}, ...$).

2. La capacidad del modelo para discriminar entre ítems previamente aprendidos y aquellos que nunca había visto disminuía a medida que la red aprendía más información. Este comportamiento es antitético a cómo opera la mente humana, donde, generalmente, a medida que aprendemos más, somos más adeptos a distinguir entre información nueva y la ya conocida. Para abordar este problema, Ratcliff intentó introducir *nodos de respuesta* diseñados para responder selectivamente a inputs tanto nuevos como antiguos. Sin embargo, esta solución no tuvo éxito, ya que estos nodos comenzaron a activarse indiscriminadamente ante cualquier tipo de input. De manera similar, un modelo que intentaba utilizar patrones de contexto tampoco logró mejorar la discriminación entre inputs antiguos y nuevos.

2.4. Soluciones

A continuación se enumerarán algunas de las soluciones propuestas a lo largo de los años para enfrentar la interferencia catastrófica en las redes neuronales, categorizadas por su taxonomía. Cabe recalcar que la taxonomía presentada acá puede diferir de otras indicadas en otros trabajos relacionados. Ni esas taxonomías ni la del presente trabajo son erradas sino simplemente un enfoque diferente al estudiar las soluciones

2.4.1. Repetición

Las técnicas adosadas a la repetición buscan almacenar muestras de ejemplos vistos durante el entrenamiento, ya sean tal como vienen en el conjunto de datos o bien muestras procesadas. Estas muestras son reutilizadas cuando el modelo está siendo entrenado con nuevos ejemplos.

Gradient Episodic Memory (2017, 2022)

Desarrollado por Lopez-Paz y otros [Lopez-Paz y Ranzato, 2017], Gradient Episodic Memory (GEM) es un método que aborda el problema de la interferencia catastrófica recordando y utilizando experiencias pasadas. Específicamente, almacena ciertos ejemplos de tareas anteriores en una ”memoria episódica”. Cuando se entrena en una nueva tarea, GEM calcula el gradiente para esa tarea y ajusta el modelo de manera que los gradientes para tareas anteriores almacenadas en la memoria episódica no se vean aumentados. Este método busca encontrar un compromiso entre la tarea actual y las anteriores, garantizando que el aprendizaje de una nueva tarea no tenga un impacto negativo en las ya aprendidas.

Una variación interesante de *GEM* es *Averaged GEM (A-GEM)*, estrategia que dice ser igual o mejor que *GEM* presentando la eficiencia de la que *EWC* goza respecto a dimensiones computacionales [Chaudhry *et al*., 2018]

Incremental Classifier and Representation Learning (2017)

iCaRL [Rebuffi *et al.*, 2017] es una propuesta que busca lograr un buen desempeño para problemas con una cantidad indefinida de clases a discriminar sin guardar ejemplos de entrenamiento que lo ayuden a *recordar* posteriormente, es decir, toma en consideración que no se sabe la cantidad de clases verá durante su ciclo de vida.

iCaRL aprende clasificadores y al mismo tiempo las representaciones de las características desde un *stream* de datos.

Para la **clasificación**, la propuesta se basa en conjuntos ($P_1, P_2, \cdot P_t$) de *ejemplos típicos (exemplars)* que selecciona dinámicamente del *stream* de datos. Hay exactamente un conjunto de ejemplos típicos por cada clase vista hasta ese momento del entrenamiento y además el número de ejemplos no puede exceder un parámetro K.

Para el **entrenamiento**, se procesan *batches* de clases al mismo tiempo usando una estrategia *class-incremental* y cada vez que aparece un dato que pertenece a una clase no observada, *iCarL* llama a un procedimiento interno que actualiza su conocimiento (los pesos de la red y sus ejemplos típicos).

La **arquitectura** usada por *iCaRL* es una red convolucional. Los autores de la propuesta interpretan esta red como un *feature extractor* entrenable $\varphi : \mathcal{X} \to \mathbb{R}^d$, seguido de una capa de clasificación con tantos nodos sigmoides como clases observadas hasta ese momento.

2.4.2. Regularizadores

Las técnicas que utilizan regularizadores como principal herramienta para enfrentar el fenómeno en estudio evitan guardas muestras de los ejemplos vistos durante el entrenamiento, ya sea por priorizar la privacidad de los datos o evitar una demanda excesiva de almacenamiento.

En lugar de lo anterior, prefieren añadir términos extras en la función de pérdida a minimizar, consolidando el conocimiento previo cuando aprenden en un nuevo conjunto de datos.

Elastic Weight Consolidation (2017)

kirkpatrick propuso *Elastic Weight Consolidation (EWC)* [He y Jaeger, 2018], un método para entrenar de forma secuencial una simple red neuronal en múltiples tareas.

Esta técnica supone que hay pesos que tienen mayor importancia que otros sobre ciertas tareas aprendidas previamente. Durante el entrenamiento sobre una nueva tarea, los cambios de los pesos de la red se hacen menos probables cuanto mayor sea su importancia.

Para estimar la importancia de un peso en la red, *EWC* utiliza mecanismos probabilísticos, en particular la matriz de información de Fisher para determinar cuales son los pesos más relevantes para la tarea previamente aprendida. Con esta información, *EWC* penaliza el cambio sobre parámetros relevantes para otras tareas dándole prioridad a parámetros aún no utilizados.

Fundamentos

Estadísticamente, optimizar los valores de los parámetros $\boldsymbol{\theta}$ es igual a encontrar sus valores más probables dado un conjunto de datos $\mathcal{D}$. Se puede calcular esta probabilidad condicional $p(\boldsymbol{\theta}, \mathcal{D})$ desde la probabilidad a priori de los parámetros $p(\boldsymbol{\theta})$ y la probabilidad del conjunto de datos $p(\mathcal{D}|\boldsymbol{\theta})$ utilizando la regla de Bayes:

$$\log\left(p(\boldsymbol{\theta}|\mathcal{D})\right) = \log\left(p(\mathcal{D}|\boldsymbol{\theta})\right) + \log\left(p(\boldsymbol{\theta})\right) - \log\left(p(\mathcal{D})\right)$$

La log probabilidad del conjunto de datos dado los parámetros $\log\left(p(\mathcal{D}|\boldsymbol{\theta})\right)$ es el valor negativo de la función de pérdida $-\mathcal{L}(\boldsymbol{\theta})$. Asumiendo que los datos se puede dividir en 2 conjuntos independientes, uno definiendo la tarea A ($\mathcal{D}_A$) y la otra tarea B ($\mathcal{D}_B$)

$$\log\left(p(\boldsymbol{\theta}|\mathcal{D})\right) = \log\left(p(\mathcal{D}_B|\boldsymbol{\theta})\right) + \log\left(p(\boldsymbol{\theta}|\mathcal{D}_A)\right) - \log\left(p(\mathcal{D}_B)\right)$$

Notar que el lado izquierdo sigue describiendo la distribución a posteriori de los parámetros dado el conjunto de datos entero, mientras que el lado derecho solo depende de la función de pérdida para la tarea B $\log\left(p(\mathcal{D}_B|\boldsymbol{\theta})\right)$. Esta probabilidad a posteriori debe contener la información de los parámetros que fueron importantes para la tarea A y por lo tanto es la clave para la implementación de *EWC*.

La verdadera probabilidad a posteriori es intratable, pues en la práctica no sé conoce cómo se distribuyen los datos y utilizar técnicas que traten de encontrar la verdadera distribución o al menos aproximarla tiene un costo computacional considerable, más aún tomando en cuenta que generalmente uno se encuentra trabajando con miles de datos, donde cada dato tiene más de una dimensión, por lo tanto se aproxima la posteriori como una distribución gaussiana donde la media es dada por los parámetros $\boldsymbol{\theta}^*_A$ y una diagonal de precisión dada por la diagonal de la matriz de información de Fisher F. F tiene tres propiedades importantes:

1. Es equivalente a la segunda derivada de la pérdida cerca de un mínimo
2. Puede ser computada desde las derivadas de primer orden solas y por lo tanto fácil de calcular incluso para modelos grandes

3. Está garantizado ser definida semipositiva

$$\mathcal{L}(\boldsymbol{\theta}) = \mathcal{L}_B(\boldsymbol{\theta}) + \frac{\lambda}{2} \left\| \text{diag}(\mathcal{I}(\boldsymbol{\theta}_A)) \odot (\boldsymbol{\theta} - \boldsymbol{\theta}_A) \right\|_2^2 \tag{4}$$

2.4.3. Transferencia

Las estrategias cuya base es la transferencia de conocimiento toman el conocimiento adquirido por una red que ha sido entrenada en una tarea similar, cuyo resultado se encuentra almacenado en los pesos de la red para utilizarlo para aprender nuevas tareas similares. La justificación es que las redes entrenadas ya han sido moldeadas a las estructuras y patrones del dominio de las tareas aprendidas y ya no necesita una gran cantidad de esfuerzo para inferir estas regularidades.

Pretrained networks (1993)

McRae y Hetherington [McRae y Hetherington, 1993] argumentaron que, a diferencia de las redes neuronales, los humanos no aprendemos información configurando pesos aleatorios en nuestro cerebro, sino que más bien el ser humano tiende a darle un gran valor a la información conocida a priori y esto ayuda a que se disminuya la interferencia. Ellos mostraron que cuando una red es pre-entrenada con un conjunto de datos a priori, entonces el entrenamiento secuencial posterior que debe tomar en cuenta los patrones a priori generados por el entrenamiento inicial restringen a la red en cómo aprende nueva información y la añade. Esto ocurre porque si se usa una muestra de datos con una gran estructura interna, como un idioma (Inglés, Español, ...) por ejemplo, el entrenamiento naturalmente tenderá a capturar los patrones y/o regularidades encontrados en el dominio de los datos. Como el dominio está basado en grandes regularidades, entonces la nueva información también la posee y por lo tanto tenderá a ser similar a la información previa lo que permite a la red añadir esta información sin tanta interferencia

Latent Learning (2015)

Latent learning es una técnica usada por *Gutstein* y *Stump* [Gutstein y Stump, 2015] para mitigar la aparición de la interferencia catastrófica tomando las ventajas que trae la técnica de *Transfer learning*. Esta técnica trata de encontrar una codificación óptima para cualquier nueva clase que se deba aprender, así estas nuevas son menos propensas a causar una interferencia catastrófica en las clases previamente aprendidas. Dada una red que ha aprendido a discriminar un conjunto de clases usando *ECOC* (*Error Correcting Output Codes*), es decir, lo opuesto a *One Hot Codes*, la codificación óptima para las nuevas clases son escogidas

observando las respuestas promedios de la red mientras entrena sobre el conjunto original sin haber sido expuesta a las nuevas clases, y esta codificación es referenciada como *Latently Learned Encodings*. Esta terminología toma prestada conceptos de Latent Learning, pues usa transfer learning haciendo las respuestas de la red frente a las nuevas clases tan consistentes como se pueda con las clases existentes

2.4.4. Optimizador

La familia de técnicas asociadas a optimizadores buscan reducir la interferencia entre múltiples tareas manipulando la forma de aprender del algoritmo modificando el funcionamiento del optimizador de parámetros de la red. Algunas de las alteraciones recaen sobre el gradiente proporcionado por el propio optimizador.

Novelty Rule (1990)

Kortge [Kortge, 2022], con el fin de alivianar el efecto de la interferencia propuso una regla llamada regla de novelty. Como su nombre lo indica, esta técnica ayuda a que la red solo deba aprender los componentes de los nuevos inputs que difieran de los viejos, de esta forma, la red solo cambia los pesos que no fueron previamente dedicados a almacenar información y por lo tanto reduciendo la sobreposición en la representación dentro de las capas ocultas. Para aplicar esta técnica, durante el entrenamiento los vectores de entrada son reemplazados por un vector que representa a los componentes que difieren (novelty vector). Cuando se utiliza esta técnica en redes estándar que usan backpropagation, se nota muy poca o nada de interferencia cuando se hace entrenamiento secuencial, sin embargo, una limitación que presenta esta técnica es que debe usarse con auto-encoders o redes auto asociativas.

Orthogonal Gradient Descent (OGD) (2019)

El método *Orthogonal Gradient Descent (OGD)* [Farajtabar *et al.*, 2020] fue propuesto para mitigar la interferencia catastrófica restringiendo la dirección de los gradientes. Durante el entrenamiento secuencial, *OGD* ajusta la dirección del gradiente actual de manera que sea ortogonal a los gradientes de tareas previas. Este método aprovecha la ortogonalidad para garantizar que el aprendizaje en la nueva tarea tenga un impacto mínimo en las tareas previamente aprendidas. Al hacerlo, *OGD* reduce la interferencia catastrófica y mejora la retención del conocimiento anterior al aprender nuevos datos o tareas.

Fundamentos

Se denota al gradiente de la función de pérdida para una tarea B como g, al ortogonalizar g respecto a un conjunto de vectores a definir, se obtiene $\tilde{g}$.

$$\tilde{g} \perp \nabla f_j(x; w), \forall x \in T_A, j = 1, 2, 3, \ldots, c \tag{5}$$

Al mover $\tilde{g}$ progresivamente en esa dirección, se logra minimizar cambios en las predicciones sobre una tarea A, así utilizamos la capacidad del modelo de forma eficiente. Sin embargo, dado que en *continual learning* no se tiene acceso a los datos de tareas anteriores, no podemos calcular $\nabla f_j(x; \boldsymbol{\theta})$ para datos de la tarea A mientras obtenemos $\tilde{g}$ para la tarea B, así que aprovechando la sobreparametrización de las redes neuronales, en la vecindad de w óptimo para la tarea A, se puede hallar un óptimo tanto para la tarea A como la tarea B. ¿Qué implica esto?, $f(x; \boldsymbol{\theta}) \approx f(x; \boldsymbol{\theta}^*_A)$ y por lo tanto se puede utilizar este valor como aproximación.

$$\tilde{g} \perp \nabla f_j(x; \boldsymbol{\theta}^*_A), \forall x \in T_A, j = 1, 2, 3, \ldots, c \tag{6}$$

Cabe destacar que $\tilde{g}$ también es una dirección descendiente, y por lo tanto $\exists \varepsilon > 0, \forall \eta : 0 < \eta < \varepsilon$, dar un paso $\eta\tilde{g}$ reduce la pérdida.

Algorithm 2 Orthogonal Gradient Descent

Input Task Sequence $T_1, T_2, T_3, \ldots$, Learning Rate η
Output Optimal parameter w

$S \leftarrow \{\}; w \leftarrow w_0$
for TaskID $k = 1, 2, 3, \ldots$ **do**
  repeat
    $g \leftarrow$ Stochastic/Batch Gradient for T_k at w
    $\tilde{g} \leftarrow g - \sum_{v \in S} \text{proj}_v(g)$ {Project g on v}
    $w \leftarrow w - \eta\tilde{g}$
  until convergence
  for $(x, y) \in T_k$ and $k \in [1, c]$ s.t. $y_k = 1$ **do**
    $u \leftarrow \nabla f_k(x; w) - \sum_{v \in S} \text{proj}_v(\nabla f_k(x; w))$
    $S \leftarrow S \cup \{u\}$
  end for
end for
return w

2.4.5. Alteración de entrada

El conjunto de técnicas que caen en esta categoría se caracterizan por adulterar de alguna forma u otra los vectores de entrada para mejorar el desempeño del modelo. No son simplemente un pseudónimo para *data augmentation* sino que adulteraciones sobre la entrada que buscan alterar el comportamiento de los parámetros de la red.

Node Sharpening (1991)

El suavizado de nodos es una técnica para mejorar la generalización y la precisión agregando ruido a las entradas de las neuronas. El ruido hace que la red sea más resistente a los cambios en los datos de entrada y ayuda a evitar que aprenda patrones específicos de los datos de entrenamiento.

El suavizado de nodos se puede realizar de varias maneras, pero una forma común es agregar ruido Gaussiano a las entradas de las neuronas. El ruido Gaussiano es un tipo de ruido aleatorio que tiene una distribución normal. Al agregar ruido Gaussiano a las entradas de las neuronas, se hace que la red sea menos sensible a los cambios en los datos de entrada y ayuda a evitar que aprenda patrones específicos de los datos de entrenamiento.

El suavizado de nodos ha demostrado ser eficaz para mejorar la generalización y la precisión en una variedad de redes neuronales artificiales. Es una técnica simple pero efectiva que puede ayudar a mejorar el rendimiento de las redes neuronales artificiales en una variedad de tareas [French, 1991].

Ortogonalización (1995)

Muchas de las primeras técnicas acerca de reducir la superposición implica hacer que el vector de entrada o los patrones de activación de las unidades ocultas sean ortogonales entre sí. *Lewandowsky* y *Li* [Lewandowsky y Li, 1995] notaron que la interferencia es menor en los patrones aprendidos de forma secuencial si es que los vectores de entrada de estos son ortogonales entre sí. Una de las técnicas que pueden crear representaciones ortogonales en las capas ocultas implica hacer el valor de las features bipolares (rango entre -1 y 1 en lugar de 0 y 1). Los patrones ortogonales tienden a reducir la interferencia entre sí pero no la elimina del todo e incluso otro tipo de problemas siguen presentando interferencia en el aprendizaje a pesar de incluir este tipo de técnicas.

Otras formas de ortogonalización abordan regularizadores (tal como lo hace *TensorFlow.Keras*) y *ortogonalización* en el proceso de optimización (ver método *OGD*).

2.4.6. Otros

Learning without Forgetting

El aprendizaje sin olvido (*LwF*) [Li y Hoiem, 2017] es un método que busca reemplazar técnicas de *fine-tuning*, *multitasking*, *feature extraction* y *joint learning*.

Los puntos claves de *LwF* son:

- **Preserva el rendimiento en las tareas anteriore**s. *LwF* es capaz de mantener el rendimiento en las tareas anteriores incluso sin acceso a los datos de entrenamiento originales. Esto es importante para aplicaciones en las que el acceso a los datos de entrenamiento originales es limitado o costoso.
- **Mejora el rendimiento de la nueva tarea**. *LwF* a menudo mejora el rendimiento de la nueva tarea en comparación con otros métodos de aprendizaje continuo. Esto se debe a que *LwF* utiliza la pérdida de tarea antigua para regularizar el modelo, lo que ayuda a evitar el sobreajuste.
- **Es eficiente computacionalmente**. *LwF* es más eficiente computacionalmente que otros métodos de aprendizaje continuo, como el entrenamiento conjunto. Esto se debe a que *LwF* no requiere entrenar el modelo en los datos de todas las tareas a la vez.

Maximally Interfered Retrieval (MIR)

MIR [Rahaf y Lucas, 2019] es una técnica de repetición de recuerdos que se centra en los ejemplos pasados que son más susceptibles de ser interferidos por los nuevos datos. Para ello, *MIR* utiliza una función de interferencia para medir la probabilidad de que un ejemplo pasado sea interferido por un nuevo ejemplo.

La función de interferencia puede definirse de varias maneras. Una forma común es utilizar una función de similitud entre los ejemplos pasado y nuevo. Por ejemplo, si los ejemplos pasado y nuevo son muy similares, es probable que se interfieran entre sí.

Una vez que se ha calculado la función de interferencia para todos los ejemplos pasados, *MIR* selecciona los ejemplos con la mayor interferencia. Estos ejemplos se utilizan para entrenar el modelo, con el objetivo de prevenir el olvido catastrófico.

Estas soluciones propuestas tienen el objetivo de abordar uno de los problemas más desafiantes en el entrenamiento de redes neuronales: la interferencia catastrófica. Mientras que algunas técnicas, como *Novelty Rule* o *Node Sharpening*, se centran en ajustar el proceso de aprendizaje en sí, otras, como *Latent Learning* o *EWC*, tratan de preservar el conocimiento anterior incorporando memoria o restricciones adicionales en la actualización de los pesos.

Cada una de estas técnicas tiene sus propias ventajas y limitaciones, pero todas buscan mejorar la capacidad de las redes neuronales para aprender de manera secuencial sin olvidar información previamente adquirida.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

En el presente capítulo se describen propuestas desarrolladas con el fin de solventar el problema introducido en el capítulo uno de este trabajo. Se introducen 3 propuestas y sus variantes directas, junto a una breve explicación sobre cómo intentan apaciguar los efectos producidos por la interferencia catastrófica sobre el modelo base.

Todas las propuestas presenten son categorizadas como estrategias regularizadoras de parámetros ya que su propuesta de valor reside en la modificación de la función de pérdida.

3.1. Propuesta con Información de Fisher

Inspirado en la estrategia presentada en el capítulo anterior, *Elastic Weight Consolidation (EWC)*, esta propuesta tiene por objetivo principal restringir la variabilidad en los parámetros de la red que fueron estimados como importantes a la hora de realizar una inferencia correcta sobre una tarea aprendida. La manera de restringir los pesos difiere de *EWC* en cómo es utilizada la importancia de cada peso θ_i.

La función de pérdida base utilizada en esta propuesta se presenta en la ecuación (7)

$$\mathcal{L} = \mathcal{L}_c + \lambda_1 \left\| \boldsymbol{u} \odot (\boldsymbol{\theta} - \boldsymbol{\theta}^*_A) \right\|_1 + \lambda_2 \left\| (\mathbf{1} - \boldsymbol{u}) \odot (\boldsymbol{\theta} - \boldsymbol{\theta}^*_A) \right\|_2^2 \tag{7}$$

3.1.1. ¿Cómo resuelve el problema?

Esta ecuación se puede descomponer en tres elementos claves para lograr el resultado deseado.

El primer componente es la función de pérdida de los datos ($\mathcal{L}_c$). Con este elemento, la red se encarga de corregir los errores en la inferencia durante el entrenamiento.

Para comprender de lleno qué hace el segundo componente, es preciso definir qué es el vector $\boldsymbol{u}$. La importancia de un parámetro θ_i se obtiene a través de la matriz de información de *fisher* $\mathcal{I}$.

$$u_i = u(i) = \begin{cases} 1, & \text{if } \mathrm{diag}(\mathcal{I})_i \geq \mu \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

En términos prácticos, se umbraliza la importancia de los pesos de la red considerando si es importante o no cuando cruza un cierto umbral cuyo valor es un hiperparámetro de la estrategia propuesta. Si el elemento i-ésimo de $\boldsymbol{u}$ es 1, significa que el peso θ_i es importante para la red cuando se clasifican los datos de una tarea.

Esto quiere decir que

$$\|\boldsymbol{u} \odot (\boldsymbol{\theta} - \boldsymbol{\theta}^*_A)\|_1 \tag{9}$$

penaliza el cambio en los parámetros que son considerados importantes para la red con una norma 1, es decir, incentivará a no alterar los pesos de la red importantes.

El último componente se encarga de penalizar el cambio de valores en los pesos que fueron considerados no importantes de una forma menos agresiva utilizando la norma $l2$

$$\|(\mathbf{1} - \boldsymbol{u}) \odot (\boldsymbol{\theta} - \boldsymbol{\theta}^*_A)\|^2_2 \tag{10}$$

Notar que como $\boldsymbol{u}$ son los pesos importantes, entonces $\mathbf{1} - \boldsymbol{u}$ son los pesos no importantes.

3.1.2. Variación

Una alternativa a cómo penaliza esta propuesta los parámetros de la red es aplicando penalización específicas a los pesos asociados a cada tarea aprendida. La ecuación (11) englosa esta idea

$$\mathcal{L} = \mathcal{L}_c + \lambda_p \cdot \sum_{i=1}^{t-1} (\lambda_1 \cdot \|\boldsymbol{U}_i \odot (\boldsymbol{\theta} - \boldsymbol{\theta}^*_i)\|_1 + \lambda_2 \cdot \|(\mathbf{1} - \boldsymbol{U})_i \odot (\boldsymbol{\theta} - \boldsymbol{\theta}^*_i)\|^2_2) \tag{11}$$

donde

- t es la tarea actual.
- U es la matriz de importancias umbralizadas, donde la j-ésima fila representa u para la j-ésima tarea aprendida.
- $\boldsymbol{\theta}^*_i$ son los parámetros obtenidos al final del entrenamiento de la i-ésima tarea.

La explicación componente a componente es equivalente a la versión *vanilla* pero orientado a tareas por separado

3.2. Propuesta con regularización sobre los pesos

El siguiente enfoque propuesto apunta a optimizar la eficiencia del modelo a través de la minimización del número de parámetros de la red realmente utilizados para realizar inferencias de exactitud considerablemente buena mientras aún permite cierta flexibilidad en los estos pesos. Esto es logrado a través de un término penalizador el cual, valga la redundancia, penaliza parámetros cuya magnitud ha superado un umbral ε (hiper-parámetro de la estrategia propuesta). El objetivo es lograr un balance entre la flexibilidad del modelo y el deseo por reducir el número de parámetros innecesarios tal como se plantea en la ecuación (12)

En la primera experiencia enfrentada por el modelo, la función de pérdida es definida de la siguiente forma

$$\mathcal{L} = \mathcal{L}_c + \lambda_1 \left\| \boldsymbol{\theta} \right\|_1 \tag{12}$$

El objetivo es minimizar la magnitud de los parámetros tanto como sea posible mientras aún se permite flexibilidad en los pesos del modelo. Si ocurre que un peso es realmente necesario para la clasificación, entonces la estrategia puede relajar la restricción en la medida que el hiper-parámetro λ_1 lo permita. Si $\lambda_1 \cdot \left\| \boldsymbol{\theta} \right\|_1$ es relativamente más grande que $\mathcal{L}_c$, entonces el modelo será menos flexible a la hora de permitir crecer en magnitud a los pesos de la red.

Una vez establecido la relevancia de los pesos luego del entrenamiento con la primera experiencia, entonces ya no necesario utilizar una penalización tan agresiva como la norma l_1 sobre todos los pesos de la red, sino que desde la segunda experiencia en adelante solo se penalizan con la norma l_1 los pesos no utilizados hasta ese momento, mientras que los pesos utilizados se siguen penalizando pero a través de la norma l_2 tal como se presenta en la ecuación (13)

$$\mathcal{L} = \mathcal{L}_c + \lambda_1 \left\| \boldsymbol{s} \odot \boldsymbol{\theta} \right\|_1 + \lambda_2 \left\| (\mathbf{1} - \boldsymbol{s}) \odot \boldsymbol{\theta} \right\|_2^2 \tag{13}$$

3.2.1. ¿Cómo resuelve el problema?

Para entender como esta propuesta puede enfrentar el fenómeno de olvido catastrófico, hay que establecer que es exactamente el vector $\boldsymbol{s}$. Este vector se define de la siguiente forma

$$s_i = s(\theta_i) = \begin{cases} 0, & \text{if } \theta_i \geq \varepsilon \\ 1, & \text{otherwise} \end{cases} \tag{14}$$

Esta ecuación quiere decir que s_i es 0 si es que el parámetro θ_i es considerado como importante para la correcta inferencia sobre la experiencia aprendida. Como su valor es 0 entonces

en la función de pérdida no se penaliza con norma l_1 dado que $s_i \cdot \theta_i = 0 \cdot \theta_i$, sin embargo si se penaliza con norma l_2 dado que $(1 - s_i) \cdot \theta_i = 1 \cdot \theta_i$.

El vector $\boldsymbol{s}$ se obtiene en tiempo de ejecución del entrenamiento, más precisamente luego de haber haberse computado las inferencias sobre los datos de entrenamiento y haber obtenido la pérdida por clases $\mathcal{L}_c$, de esta manera no se almacena nada en memoria.

3.3. Propuesta con optimización de uso de pesos

La tercera propuesta presentada en este trabajo, al igual que la propuesta anterior, tiene por objetivo optimizar la eficiencia del modelo al minimizar la cantidad de parámetros realmente utilizadas durante el proceso de clasificación. Sin embargo, a diferencia de la propuesta anterior, esta penaliza el variación de los parámetros respecto a parámetros encontrados en la experiencia anterior mas no la magnitud de cada parámetro.

El beneficio que presenta este enfoque de penalización es que permite mantener una nueva solución cercana a la vecindad, tal como la primera propuesta a la vez que penaliza los pesos no utilizados en las inferencias si sobrepasan un umbral ε (hiper-parámetro de la estrategia).

En un principio del trabajo, se propuso añadir un término extra que penalizara la cantidad de pesos usados a través de la norma 0, pero debido a que no se puede crear una función diferenciable que permita obtener la derivada de este término respecto a cualquier peso de la red y además a que añade ruido a la pérdida total $\mathcal{L}$ se descartó la idea.

La formulación que describe el espíritu de esta propuesta es la ecuación (15)

$$\mathcal{L} = \mathcal{L}_c + \lambda_e \cdot \|\boldsymbol{p}_A^* \odot (\boldsymbol{\theta} - \boldsymbol{\theta}_A^*)\|_1 + \lambda_f \cdot \|(\mathbf{1} - \boldsymbol{p}) \odot \boldsymbol{\theta}\|_1 \tag{15}$$

3.3.1. ¿Cómo resuelve el problema?

El término clave de la ecuación (15) que permite entender el funcionamiento de este enfoque es $\boldsymbol{p}$ pero ¿Qué representa $\boldsymbol{p}$?

$\boldsymbol{p} \in \mathbb{R}^P$ es un vector que representa la utilización o no de un parámetro específico de la red en la inferencia. Este vector se define de la siguiente forma:

$$p_i = p(\theta_i) = \begin{cases} 1, & \text{if } \theta_i \geq \varepsilon \\ 0, & \text{otherwise} \end{cases} \tag{16}$$

De este modo $\boldsymbol{p}_A^*$, que sigue la misma notación para elementos almacenados resultado de

un entrenamiento previo, es exactamente el vector $\boldsymbol{p}$ obtenido durante la tarea anterior. Así el segundo término de la función de penalización (ecuación (17)) apunta a preservar los parámetros actuales en una vecindad cercana a los parámetros obtenidos durante la tarea anterior.

$$\lambda_e \cdot \|\boldsymbol{p}_A^* \odot (\boldsymbol{\theta} - \boldsymbol{\theta}_A^*)\|_1 \tag{17}$$

Así mismo, en la siguiente ecuación se expresa la intención de mantener la magnitud de los parámetros no utilizados en el entrenamiento lo más bajo posible.

$$\lambda_f \cdot \|(\mathbf{1} - \boldsymbol{p}) \odot \boldsymbol{\theta}\|_1 \tag{18}$$

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

Este capitulo tiene por objetivo presentar los resultados obtenidos al implementar las propuestas presentadas en el capítulo tres del presente trabajo, realizando un análisis exhaustivo de cada propuesta y contrastando los resultados obtenidos con los que se pueden obtener con algoritmos presentados en el estado del arte de *Continual Learning*.

Todo el procedimiento realizado se puede obtener para su reproducibilidad a través de un repositorio en Github [Oyanedel, 2024].

4.1. Indicadores clave de desempeño

En esta sección se enumerarán indicadores claves que son un punto clave en la consolidación de resultados del presente trabajo. Una parte de los indicadores son propios del contexto de aprendizaje continuo, sin embargo estos no son los únicos considerados para evaluar la calidad de una estrategia en particular. Otra parte de los indicadores está estrechamente relacionado con la computación requerida para mantener a los modelos durante su ciclo de vida, en particular la etapa de entrenamiento y de ejecución en un entorno de producción, o de otra forma, etapa de inferencia. Por último los indicadores relacionados al modelo en sí, y aunque algunas métricas encapsuladas aquí podrían establecerse como métricas de computación, su definición ambigua se permite agregarlas aquí.

4.1.1. Indicadores de aprendizaje continuo

Los indicadores o métricas utilizados en este apartado son relacionados al aprendizaje continuo, estas métricas son enumeradas a continuación.

1. *Accuracy* por clase
2. *Average Accuracy (AA)*
3. *Average Incremental Accuracy (AIA)*
4. *Backward Transfer (BWT)*
5. *Forgetting Measure (FM)*

4.1.2. Indicadores de computación

Los indicadores de computación son los que tienen relación con los recursos computacionales ocupados por un ordenador al ejecutar el modelo. Estas métricas son las siguientes:

1. Tiempo usado en el entrenamiento

4.2. Conjunto de datos

El conjunto de datos utilizado para realizar *benchmarks* es *Permuted MNIST*, un *dataset* derivado de *MNIST* que contiene 60000 imágenes de dígitos escritos a mano para el entrenamiento y 10000 para realizar la evaluación de los modelos. En un comienzo, las imágenes tenían domensiones de 128x128x1, sin embargo, a medida que transcurrían los años, este mutó hasta reducir el tamaño por dato a 28x28x1 (ver Figura 3).

Figura 3: Conjunto de datos en MNIST

Permuted MNIST es una mutación de este*dataset* en donde por cada clase del conjunto original se generan 10 clases realizando permutaciones de pixeles de las imágenes, dejando un total de 100 clases (ver Figura 4)

Existe literatura en el estado del arte que consideran que *Permuted MNIST* no es un conjunto de datos con la capacidad de medir el desempeño de algoritmos de aprendizaje continuo con un alto grado de confiabilidad [Farquhar y Gal, 2018], sin embargo, esta vertiente no ha abarcado una gran aceptación por parte de la comunidad. Así, este conjunto de datos es utilizado por este trabajo por su uso en muchos de los algoritmos explorados en el estado del arte permitiendo una comparación más justa.

4.3. Arquitectura

Para realizar las evaluaciones de cada estrategia se utilizó la misma red con exactamente la misma definición proveída por el *framework Avalanche* de *ContinualAI* [Lomonaco *et al.*, 2021]. Este modelo es una red *Multi Layer Perceptron* con la estructura mostrada en la Figura 5

Una capa de entrada con 28 x 28 neuronas + 1 neurona de sesgo, dos capas ocultas de 100 neuronas cada una más su neurona de sesgo y finalmente la capa de salida con 100 *logits*, dado que el *dataset* presenta 100 clases. El modelo presenta un *drop rate* de 50 %.

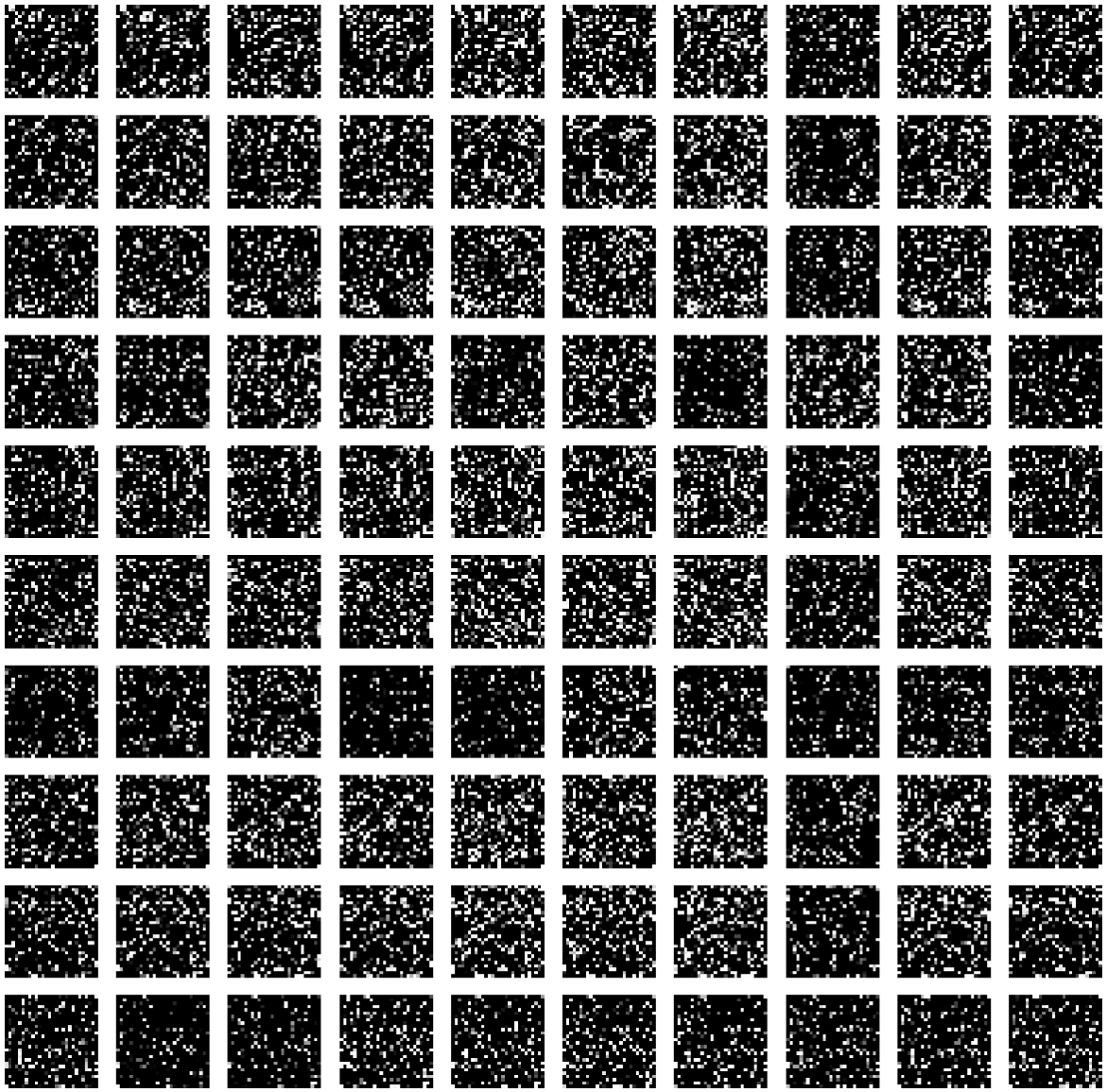


Figura 4: Conjunto de datos en PMNIST

Dada su arquitectura, en este trabajo se tienen que entrenar un total de $(784 + 1) \cdot 100 + 101 \cdot 100 + 101 \cdot 100 = 98700$ neuronas.

La razón tras la elección de la presente arquitectura es, en realidad, simple. Gran parte de los modelos del estado del arte utilizados en el *benchmark* fueron evaluados con un mismo tipo de modelo, es decir, con una red *Multi Layer Perceptron*. Si bien algunos hiper parámetros de cada modelo en particular varió en la implementación usada por cada autor, siguen utilizando el mismo tipo de red lo que es un punto clave en la investigación. Hay un par de modelos que utilizaron redes diferentes, sin embargo esos modelos no fueron contrastados en el presente trabajo para no realizar comparaciones sesgadas, dado que estos modelos pueden haber funcionado en el contexto introducido pero no necesariamente con el utilizado aquí o vice-

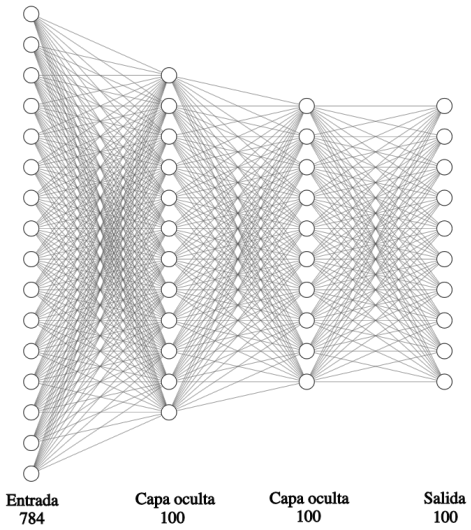


Figura 5: Arquitectura base

versa.

4.4. Proceso de evaluación

En esta sección se mide el desempeño de las estrategias basado en 3 etapas importantes. La división en etapas busca variar algún componente del entorno de trabajo manteniendo el resto de elementos fijo.

4.4.1. Comparación estándar

En este apartado se comparan las estrategias manteniendo la arquitectura de la red, y la cantidad de tareas y experiencias fijas. El objetivo es contrastar el desempeño de estrategias bajo condiciones normales.

La arquitectura utilizada en este escenario se define en la Figura 5

La cantidad de experiencias fijadas en este escenario es 10, esto significa que cada experiencia contiene 10 clases para aprender.

Accuracy

Como se puede observar en los gráficos de las Figuras 6- 15, la estrategia que posee mejor *accuracy* es *cumulative*. No es sorpresa dado que retiene los datos de experiencias previas

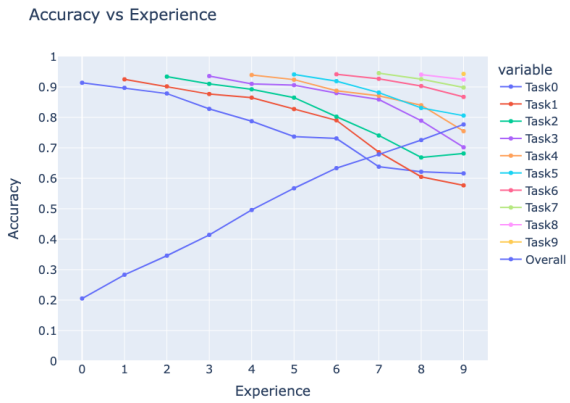


Figura 6: Accuracy de A-GEM

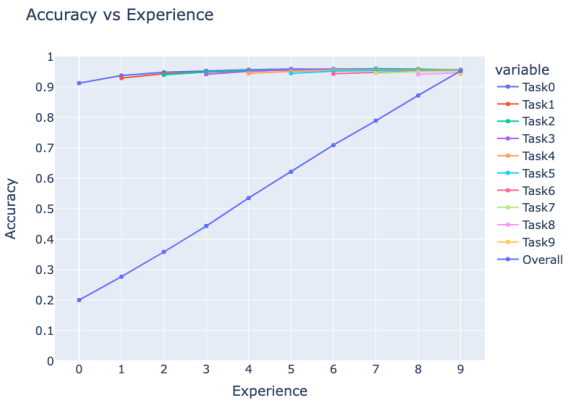


Figura 7: Accuracy de Cumulative

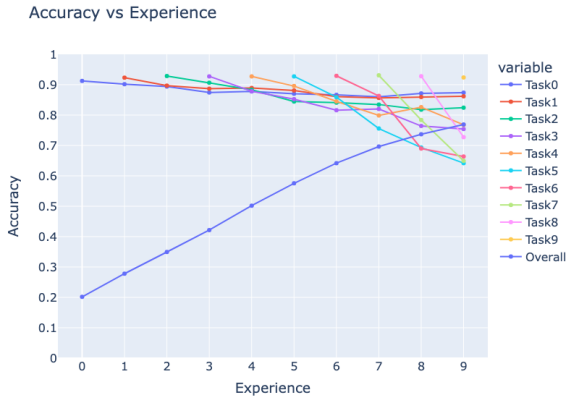


Figura 8: Accuracy de EWC

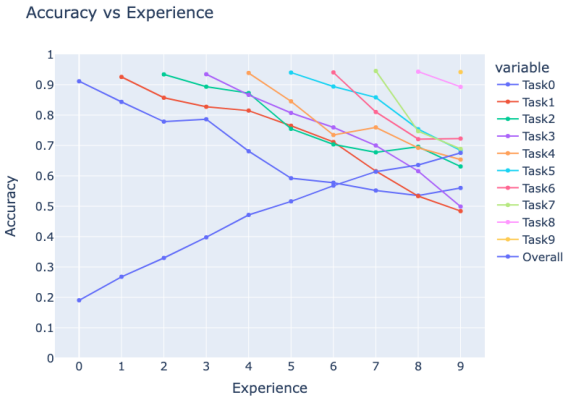


Figura 9: Accuracy de GEM

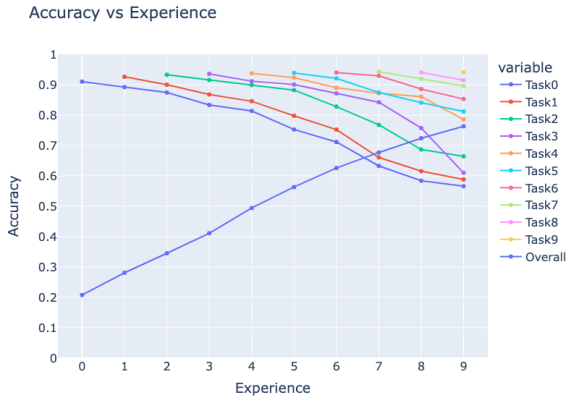


Figura 10: Accuracy de LwF

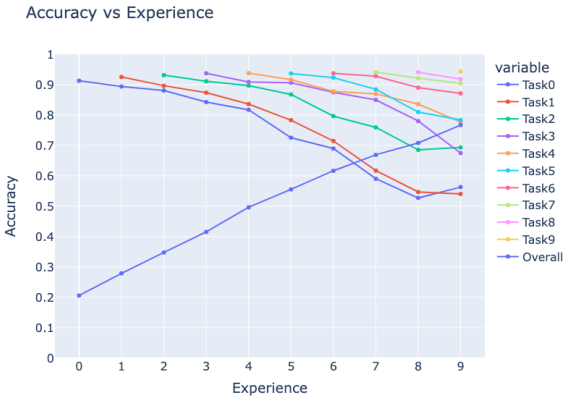


Figura 11: Accuracy de MIR

para aprender nuevas experiencias. Fuera de ello, la otra estrategia con mejores resultados es *EWC*.

EWC es capaz de mantener una buena métrica dentro de las primeras tareas a medida que se

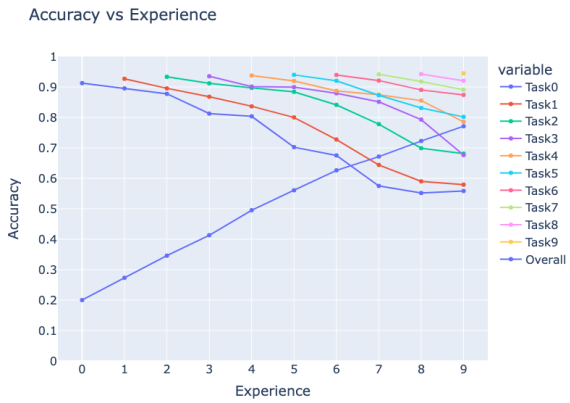


Figura 12: Accuracy de Naive

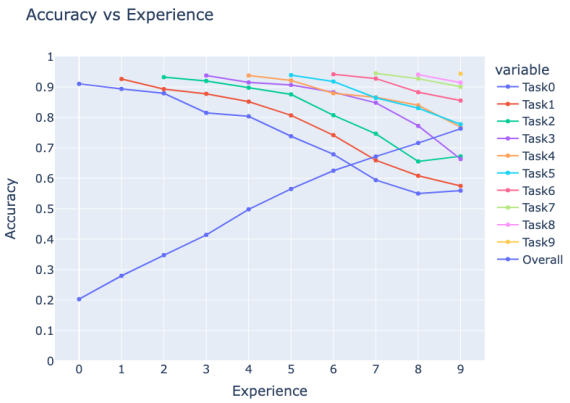


Figura 13: Accuracy de Propuesta 1

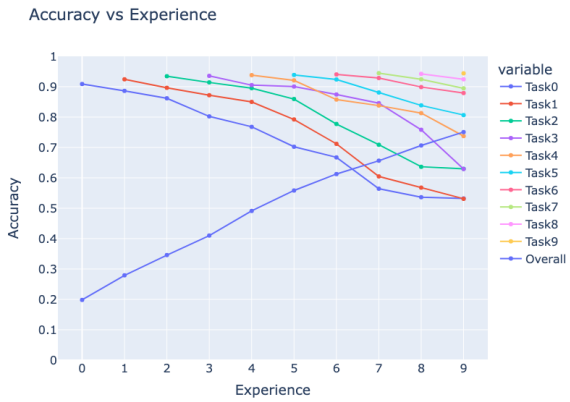


Figura 14: Accuracy de Propuesta 2

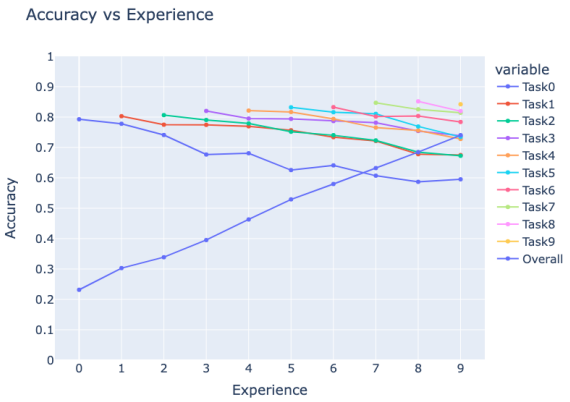


Figura 15: Accuracy de Propuesta 3

aprenden nuevas, sin embargo, se observa una menor capacidad de retención en el aprendizaje de nuevas tareas, este fenómeno se analiza en otro apartado.

La estrategia *Naive* mantiene una exactitud sorprendente dado el contexto de aprendizaje continuo, sin embargo, esto no se mantiene al aumentar la cantidad de experiencias que se deben aprender.

Las estrategias propuestas en este apartado no son capaces de superar a *EWC*. De hecho, no es capaz de aprender tanto como el resto de estrategias, sin embargo, la capacidad de retención de aprendizaje es mayor que el resto, métrica que se verá en el apartado de *forgetting measure*.

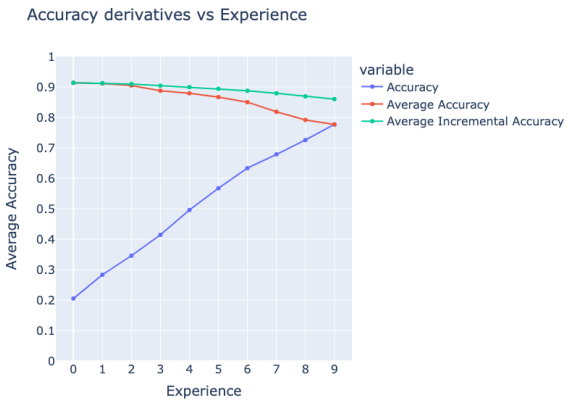


Figura 16: Average Accuracy de A-GEM

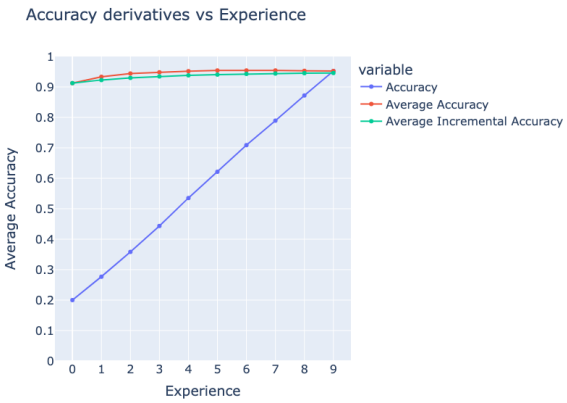


Figura 17: Average Accuracy de Cumulative

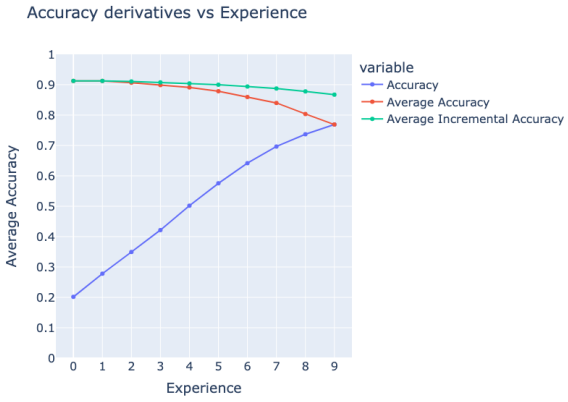


Figura 18: Average Accuracy de EWC

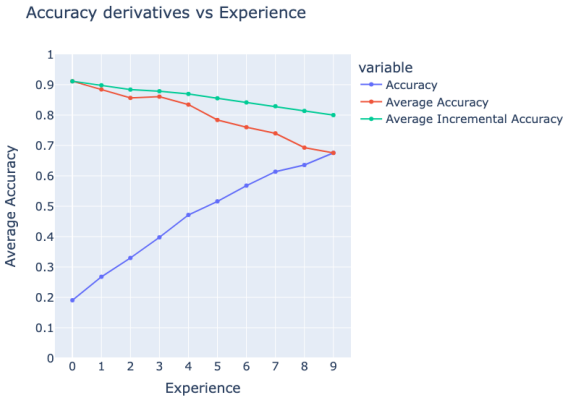


Figura 19: Average Accuracy de GEM

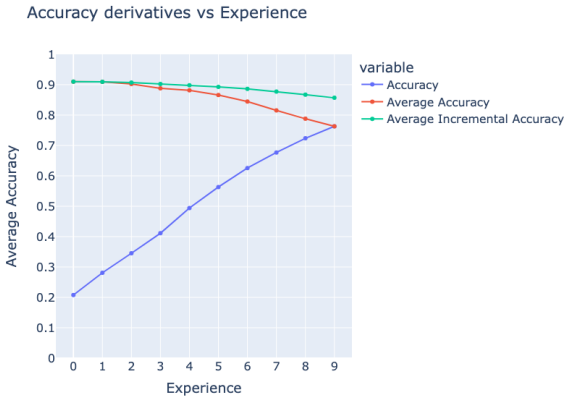


Figura 20: Average Accuracy de LwF

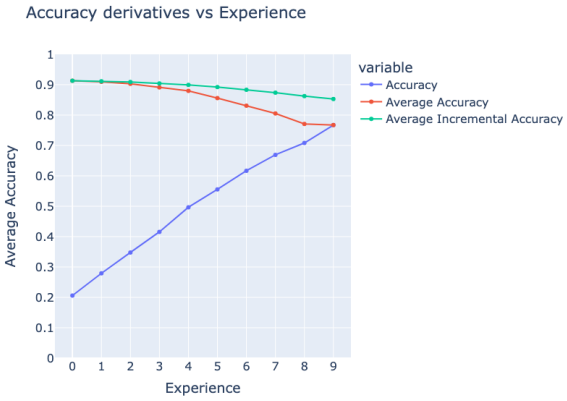


Figura 21: Average Accuracy de MIR

Derivados de Accuracy

En las Figuras 16- 25 se observan distintos *accuracies* del estado del arte resultado de cada entrenamiento de cada estrategia. Notar que todas las estrategias presentan un comporta-

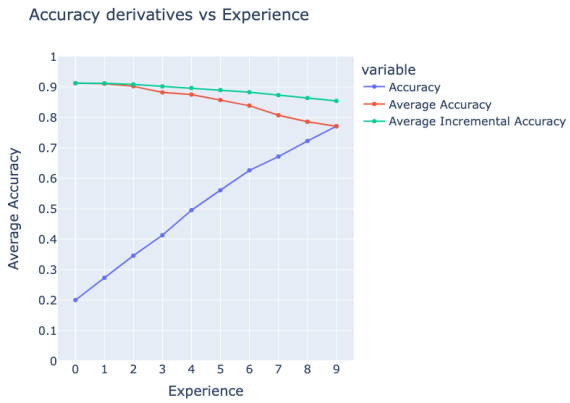


Figura 22: Average Accuracy de Naive

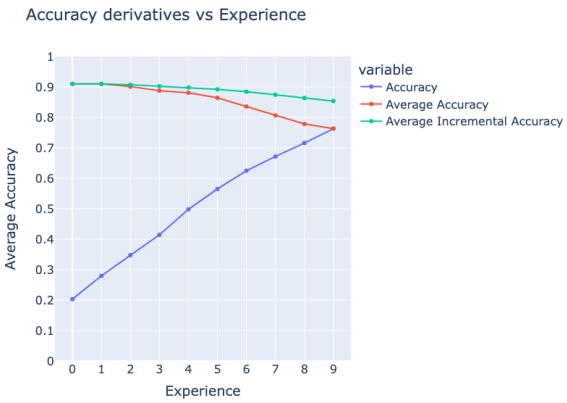


Figura 23: Average Accuracy de Propuesta 1

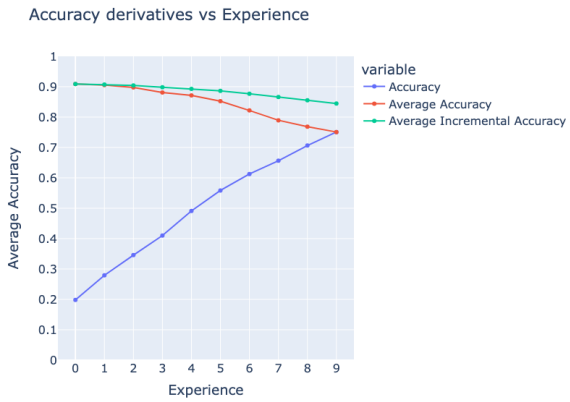


Figura 24: Average Accuracy de Propuesta 2

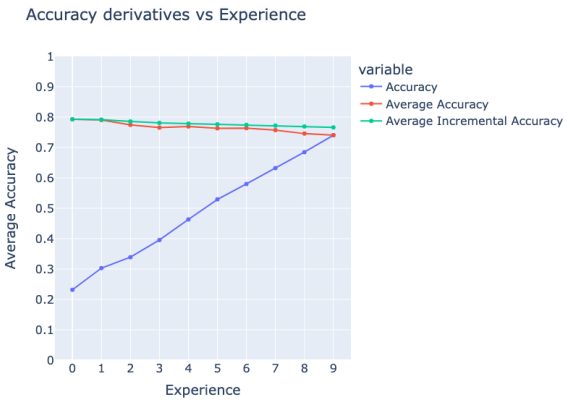


Figura 25: Average Accuracy de Propuesta 3

miento similar en cuando a su exactitud sin embargo la propuesta 3 presenta una cualidad rescatable, su *Average Accuracy* y *Average Incremental Accuracy* se mantienen bastante cerca en magnitud, esto es porque es capaz de mantener el conocimiento previo cuando se entrena con nuevas tareas.

Forgetting Measure

Lo primero que se destaca en este apartado (ver Figuras 26 a 35 es que la estrategia *cumulative* presenta métricas negativas por una razón casi lógica. Al entrenar una nueva experiencia, utiliza información de experiencias previas, lo que aumenta el desempeño por cada una de las experiencias vistas hasta ese momento.

Otro detalle importante a comentar es que todas las estrategias presentan grados de olvido en al menos una de sus tareas superior a 0.28, sin embargo, la propuesta 3 es capaz de mantener esta métrica por debajo de 0.2 en su peor tarea. Fuera de la peor tarea, el resto

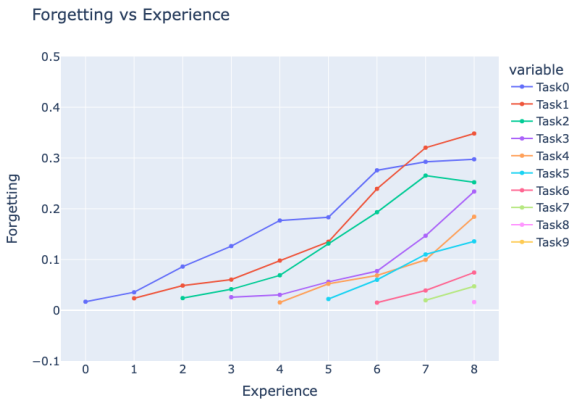


Figura 26: Forgetting de A-GEM

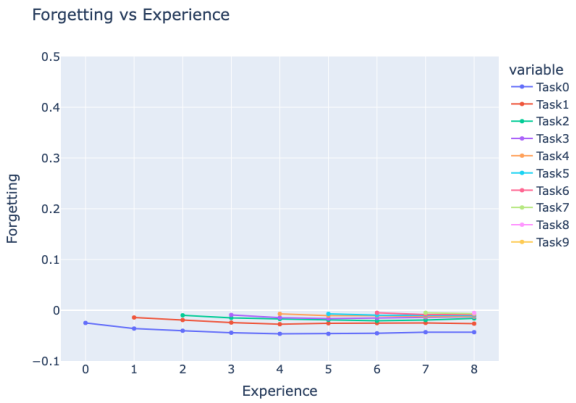


Figura 27: Forgetting de Cumulative

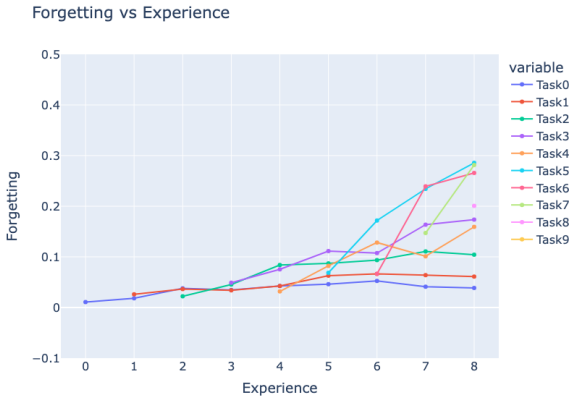


Figura 28: Forgetting de EWC

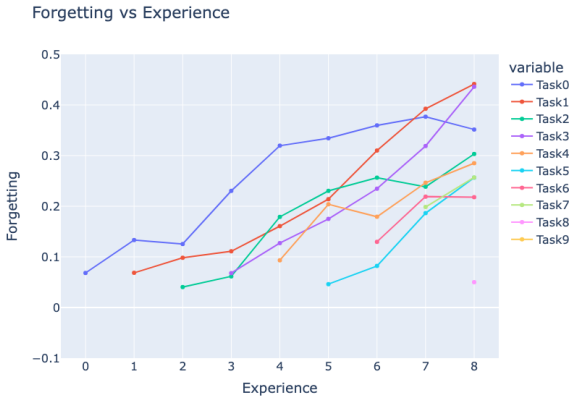


Figura 29: Forgetting de GEM

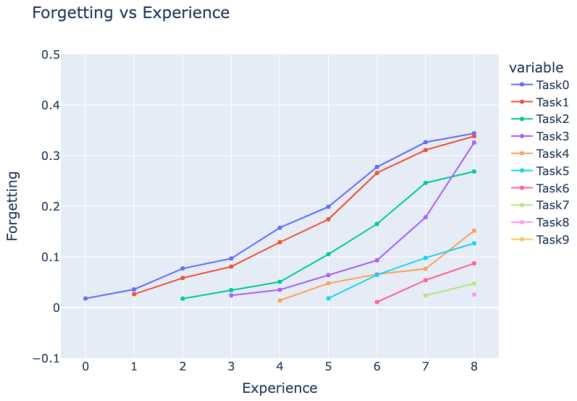


Figura 30: Forgetting de LwF

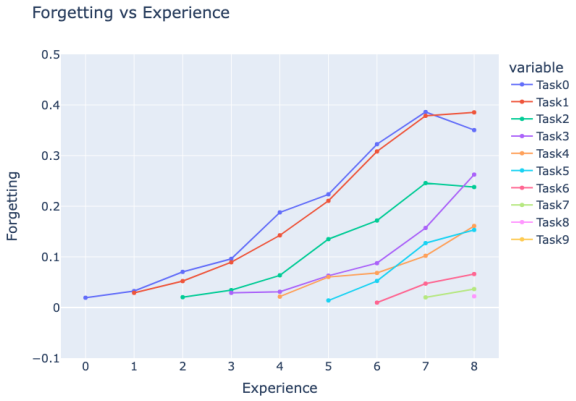


Figura 31: Forgetting de MIR

mantiene un grado de olvido considerablemente bajo respecto al resto de estrategias.

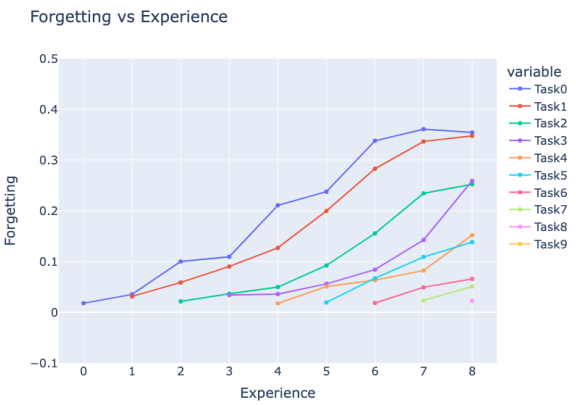


Figura 32: Forgetting de Naive

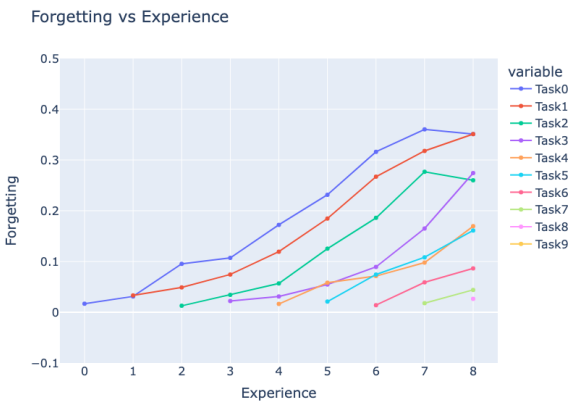


Figura 33: Forgetting de Propuesta 1

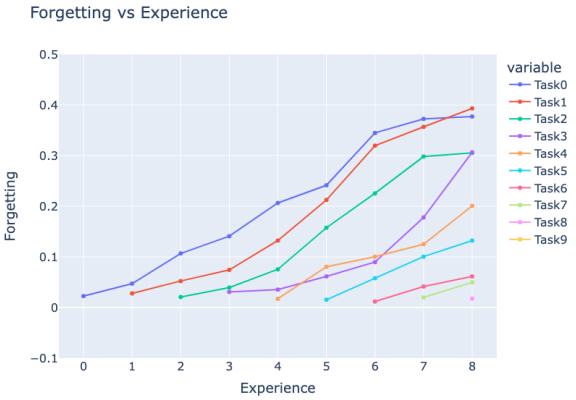


Figura 34: Forgetting de Propuesta 2

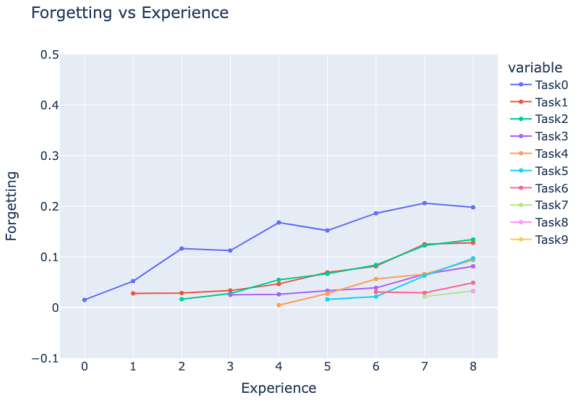


Figura 35: Forgetting de Propuesta 3

Backward Transfer

En general, en las estrategias de cumple la igualdad FM = -BWT, no obstante, esa percepción no debe guiarlos al engaño, pues la fórmula detrás de ambas métricas tiene diferencias sutiles.

Tiempo de entrenamiento

Este es un apartado muy relevante en la investigación, pues uno de los objetivos específicos es crear estrategias óptimas en rendimiento.

En la Figura 46, se aprecia que el algoritmo más lento durante el periodo de entrenamiento es *cumulative*, aunque durante las *epochs* 5 a 14, *GEM* tuvo un tiempo peor.

El resto de algoritmos se mantiene una ventana de tiempo por *epoch* cercana entre ellos.

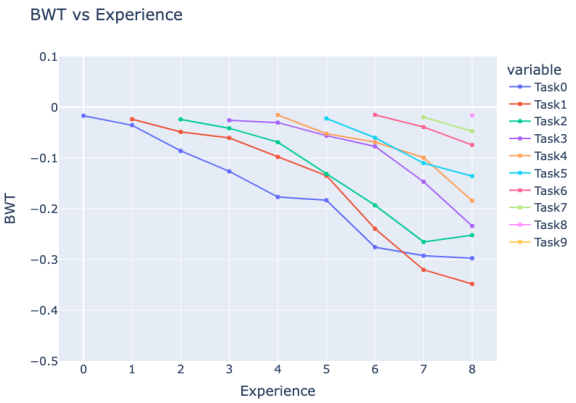


Figura 36: Backward Transfer de A-GEM

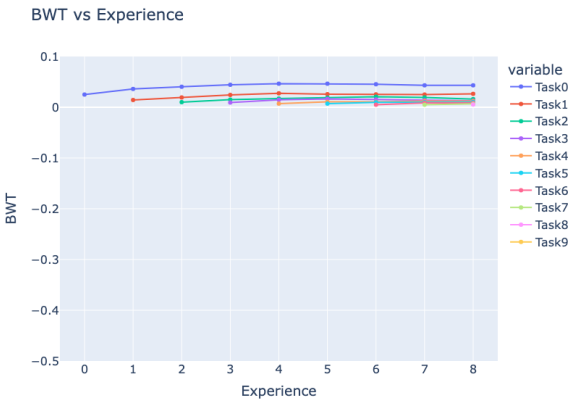


Figura 37: Backward Transfer de Cumulative

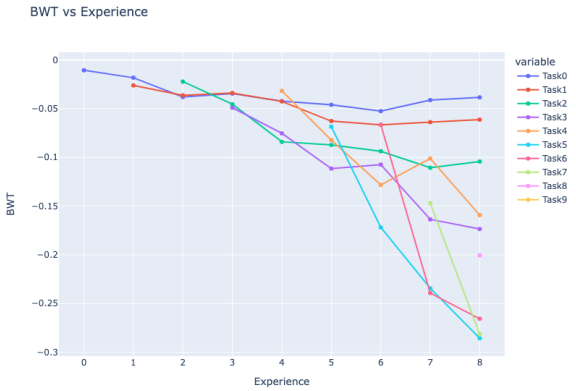


Figura 38: Backward Transfer de EWC

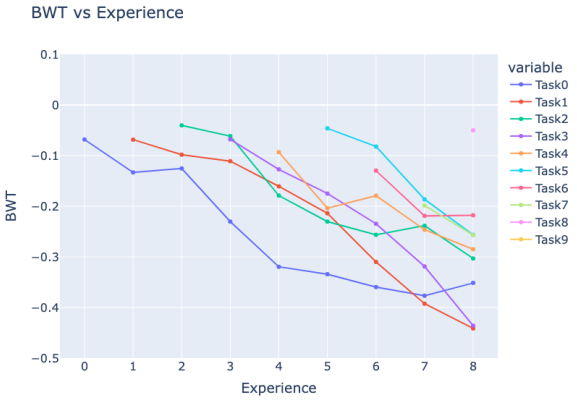


Figura 39: Backward Transfer de GEM

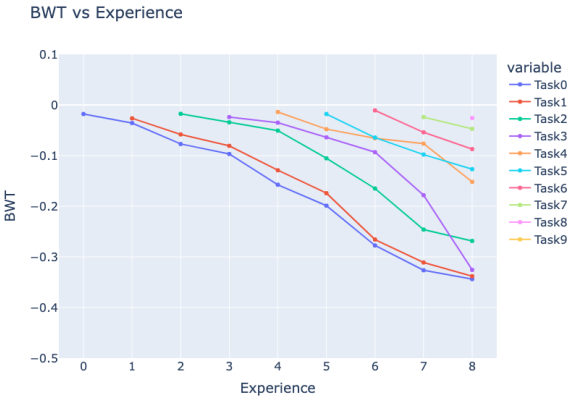


Figura 40: Backward Transfer de LwF

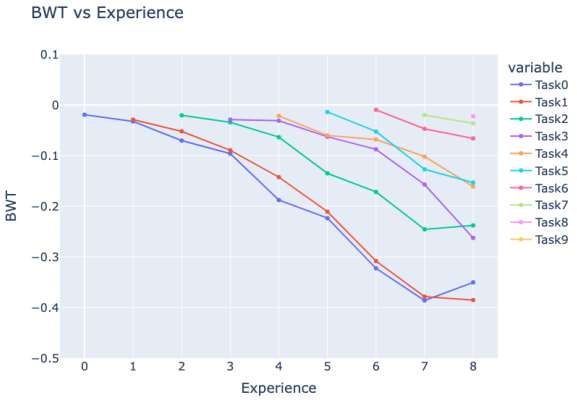


Figura 41: Backward Transfer de MIR

EWC es el más lento dentro del grupo, y naturalmente *naive* es el más veloz al no implementar sutilezas por debajo, pues todo el resto de estrategias implementan *Stochastic Gradient Descent*.

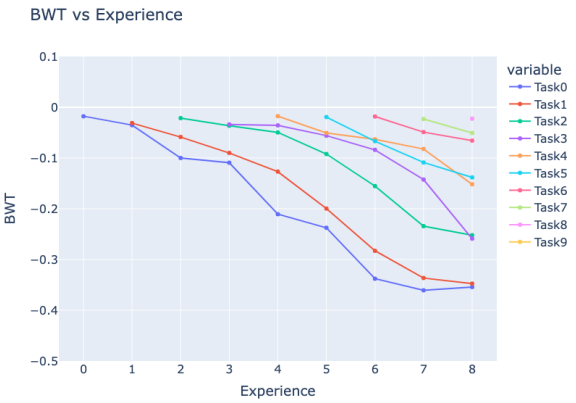


Figura 42: Backward Transfer de Naive

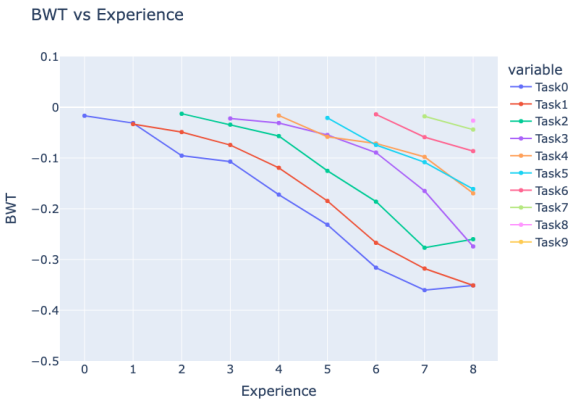


Figura 43: Backward Transfer de Propuesta 1

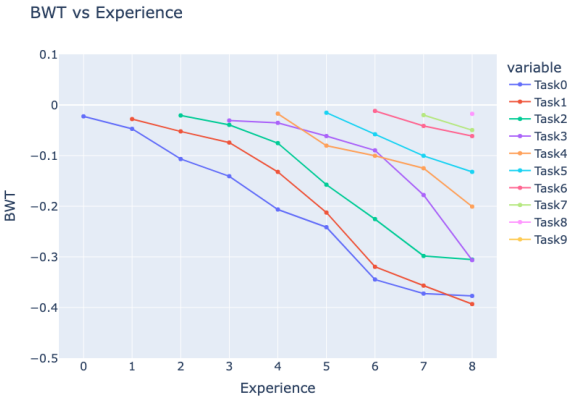


Figura 44: Backward Transfer de Propuesta 2

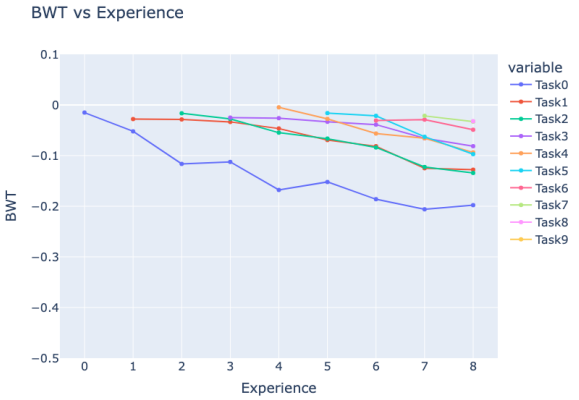


Figura 45: Backward Transfer de Propuesta 3

La tabla 1 presente un sumario de los resultados obtenidos en este apartado. En ella se añade un caso base o *baseline* cuyos valores representan *accuracies* en un entrenamiento *offline*, es decir, un entrenamiento fuera del contexto de aprendizaje continuo. Se destacan en **negrita** los mejores valores en cada tarea (No se considera ni el *baseline* ni *cumulative* por ser casos excepcionales). Naturalmente la estrategia *cumulative* ha sobrepasado los valores fijos allí, sin embargo, el motivo es porque esta última ha entrenado más veces con los datos que el modelo usado para el caso base.

La tabla 2 resume el grado de olvido por clase de cada estrategia, además de añadir el olvido promedio de cada estrategia y un promedio por tarea ponderada, es decir, considerando que las primeras tareas presentan más valores registrados que las últimas.

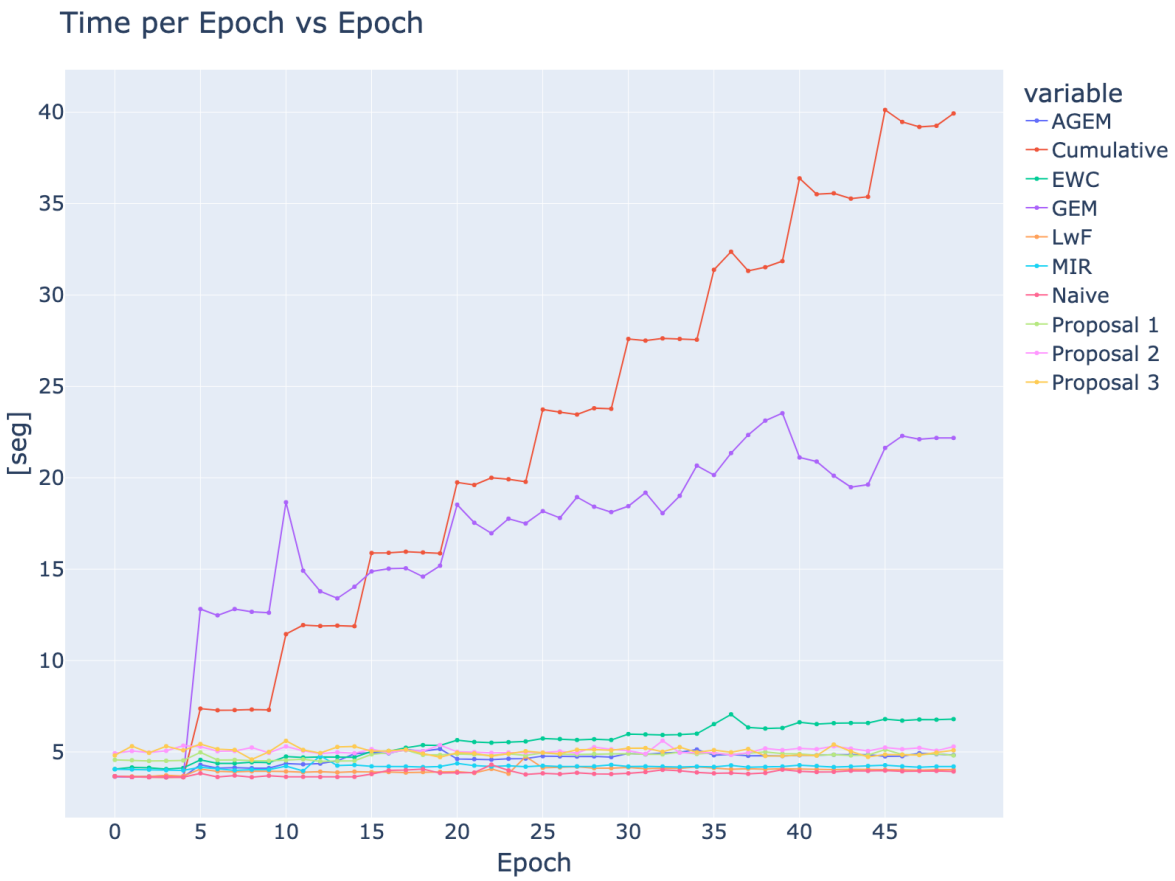


Figura 46: Tiempo por batch por cada estrategia

Tarea	1	2	3	4	5	6	7	8	9	10	General
Baseline	0,928	0,926	0,928	0,928	0,929	0,928	0,924	0,927	0,931	0,927	0,928
AGEM	0,616	0,577	0,682	0,702	0,755	0,806	0,867	0,898	**0,924**	0,943	**0,777**
Cumulative	0,956	0,956	0,956	0,954	0,957	0,955	0,952	0,952	0,947	0,942	0,953
EWC	**0,874**	**0,862**	**0,824**	**0,754**	0,768	0,642	0,664	0,649	0,727	0,924	0,769
GEM	0,560	0,484	0,631	0,499	0,653	0,684	0,723	0,689	0,893	0,942	0,676
LwF	0,566	0,588	0,664	0,610	**0,786**	**0,812**	0,853	0,896	0,915	0,941	0,763
MIR	0,563	0,540	0,694	0,675	0,777	0,784	0,872	**0,905**	0,919	0,943	0,767
Naive	0,558	0,579	0,681	0,676	0,785	0,801	0,874	0,891	0,920	**0,945**	0,771
Propuesta 1	0,559	0,575	0,672	0,663	0,768	0,777	0,855	0,901	0,914	0,943	0,763
Propuesta 2	0,532	0,531	0,629	0,629	0,738	0,807	**0,879**	0,895	**0,924**	0,944	0,751
Propuesta 3	0,595	0,675	0,672	0,739	0,728	0,735	0,784	0,814	0,819	0,842	0,740

Tabla 1: Accuracies finales por tarea y algoritmo

4.4.2. Comparación con saturación de tareas

En este apartado se comparan las estrategias manteniendo la arquitectura de la red fija pero variando la cantidad de tareas en las que se divide el conjunto de datos. El objetivo es contrastar el desempeño de estrategias cuando la cantidad de tareas es pequeña o grande.

Tarea	1	2	3	4	5	6	7	8	9	Promedio por Tarea	Promedio por tarea ponderado
AGEM	0,166	0,159	0,139	0,095	0,084	0,082	0,043	0,033	0,016	0,091	0,117
Cumulative	−0,041	−0,024	−0,017	−0,014	−0,011	−0,009	−0,007	−0,006	−0,005	−0,015	−0,020
EWC	**0,036**	**0,049**	0,078	0,113	0,101	0,190	0,190	0,214	0,201	0,130	0,098
GEM	0,255	0,225	0,187	0,227	0,202	0,143	0,189	0,228	0,050	0,189	0,209
LwF	0,170	0,173	0,127	0,120	0,071	0,077	0,051	0,036	0,026	0,094	0,121
MIR	0,188	0,199	0,130	0,105	0,083	0,087	0,041	0,028	0,022	0,098	0,129
Naive	0,196	0,184	0,120	0,102	0,073	0,083	0,044	0,037	0,022	0,096	0,125
Propuesta 1	0,187	0,175	0,136	0,106	0,083	0,091	0,053	0,031	0,026	0,099	0,127
Propuesta 2	0,207	0,196	0,160	0,117	0,105	0,076	0,038	0,035	**0,018**	0,106	0,140
Propuesta 3	0,134	0,068	**0,072**	**0,045**	**0,050**	**0,049**	**0,036**	**0,027**	0,032	**0,057**	**0,070**

Tabla 2: Forgetting promedio por tarea y algoritmo

Es importante recalcar que no se compararán todos los algoritmos presentados en la sección anterior sino que un selecto subconjunto de ellos. Estos algoritmos son:

- *Elastic Weight Consolidation*
- *Naive*
- Propuesta 1
- Propuesta 2
- Propuesta 3

Dado que en la comparación estándar ya se probaron las estrategias con 10 tareas, en esta sección se realizará el *benchmark* con 20 y 50 tareas.

4.4.3. 20 tareas

Dividir el conjunto de datos en un total de 20 tareas implica que cada tarea contiene 5 clases, cantidad de clases aún suficiente para que un modelo de aprendizaje continuo sea capaz de moldear sus componentes internos a patrones que se encuentren en las primeras tareas y así, las siguientes tareas a aprender no corrompan el conocimiento.

En la Figura 47, vemos que a la estrategia de *Elastic Weight Consolidation* le ocurre el problema evidenciado en el entrenamiento estándar con 10 tareas. Este problema es que aprende bien los primeros conjuntos de tareas, y estos influyen negativamente en el aprendizaje de tareas futuras, manteniendo un grado de olvido marginal en las primeras tareas y un olvido realmente catastrófico en las últimas tareas aprendidas.

Para la estrategia ingenua (ver Figura 48), si bien no mantiene el conocimiento de las primeras tareas con la misma fuerza que *EWC*, si es capaz de retener mejor la información de las últimas tareas lo que responde a porque el *accuracy* general es muy cercano al de la estrategia anterior.

Para las estrategias propuestas 1 y 2, el resultado mostrado en las Figuras 49 y 50 respectivamente no difiere mucho de una estrategia ingenua. Estas estrategias no han logrado diferenciarse con éxito del resto de estrategias del estado del arte.

La estrategia que ha sido una gran sorpresa es la número 3, cuyos resultados visibles en la Figura 51 evidencian como la información tanto en las primeras como últimas tareas vistas se mantienen relativamente estables a comparación del resto de algoritmos. Si bien, esta no mantiene una exactitud grande al aprender tareas, siendo vencida en esta métrica por el resto de estrategias, su cantidad de olvido por tarea es pequeño.

La tabla 3 posee las métricas consideradas en este experimento. Desde esa tabla se puede evidenciar como la estrategia propuesta #3 tiene un desempeño mejor que el resto de algoritmos de la tabla.

Métrica	Accuracy	Average Incremental Accuracy	Forgetting medio
EWC	0,581	**0,764**	0,245
Naive	0,552	0,745	0,282
Propuesta 1	0,561	0,754	0,272
Propuesta 2	0,546	0,746	0,283
Propuesta 3	**0,673**	0,732	**0,131**

Tabla 3: Métricas finales para entrenamiento de 20 tareas

4.4.4. 50 tareas

Dividir el conjunto de datos en 50 tareas con 2 clases cada una acrecenta el problema ya evidenciado en el entrenamiento con 20 tareas pero con una fuerza aún mayor dado que al tener que solamente aprender 2 clases por tarea en las primeras fases, afecta a cómo los pesos de la red son ajustados a los datos y pueden ser modificados a medida que se aprenden nuevas tareas.

En la Figura 52 se observa como, a partir de la experiencia 20 en adelante, *EWC* es incapaz de mantener el conocimiento adquirido un entrenamiento atrás para una tarea, es decir, su exactitud baja desde alrededor de 90 % a un aproximado de 20 %. También se debe destacar que su rendimiento en las primeras tareas es admirable, manteniendo un buen desempeño al clasificar clases de las primeras tareas vistas sobre un 80 %.

Para las estrategias ingenua, propuesta 1 y propuesta 2, el rendimiento en la clasificación de datos ya vistos por clases no decae drásticamente como en el caso de *Elastic Weight Consolidation*, sin embargo, al aprender una nueva tarea, el *accuracy* justo después del entrenamiento supera el umbral de 90 %, a diferencia del resto de estrategias, como se puede avalar en las Figuras 53, 54 y 56 respectivamente.

Por último, la estrategia propuesta 3, manteniendo la consistencia de sus resultados en el

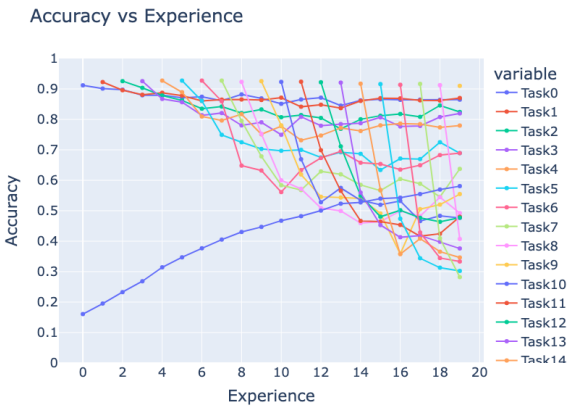


Figura 47: Accuracy en EWC con 20 tareas

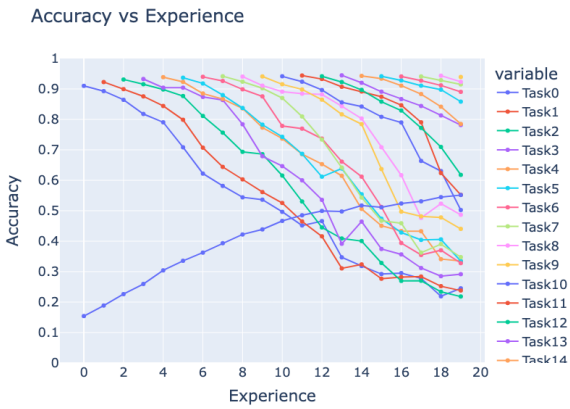


Figura 48: Accuracy en Naive con 20 tareas

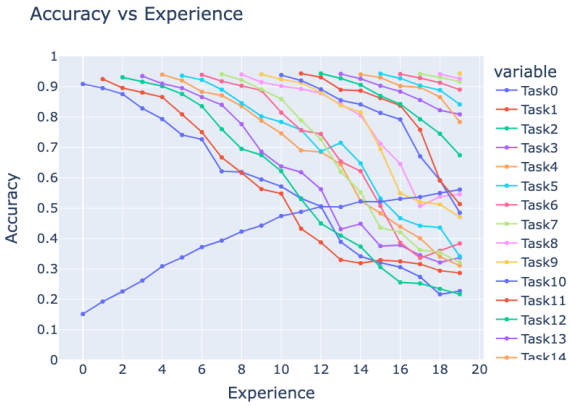


Figura 49: Accuracy en Propuesta 1 con 20 tareas

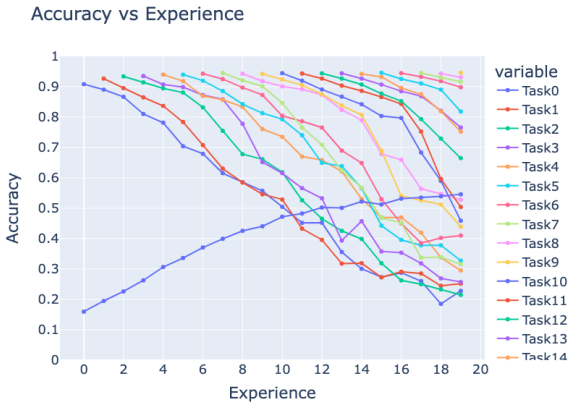


Figura 50: Accuracy en Propuesta 2 con 20 tareas

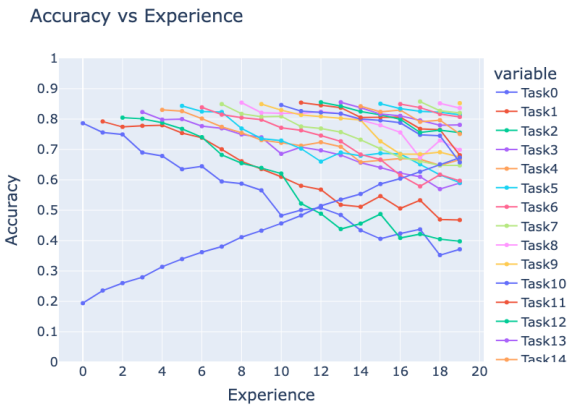


Figura 51: Accuracy en Propuesta 3 con 20 tareas

entrenamiento con 20 tareas, ahora muestra su capacidad de retener información por un

gran periodo de tiempo. Naturalmente, a medida que transcurre el entrenamiento con nuevas tareas, las tareas antiguas van siendo olvidadas pero con un decaimiento promedio más bajo que todo el resto de estrategias. Al igual que *Elastic Weight Consolidation*, la propuesta 3 no baja su métrica de exactitud bajo el 10 %.

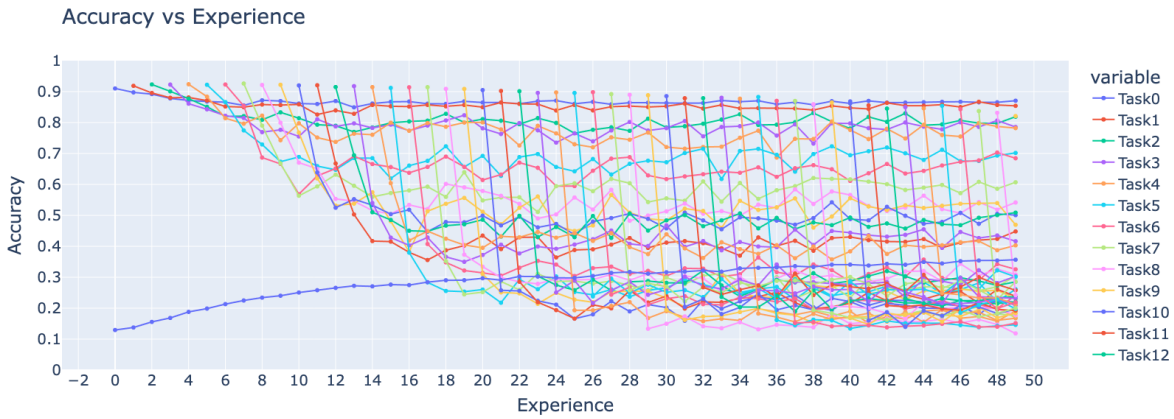


Figura 52: Accuracy en EWC con 50 tareas

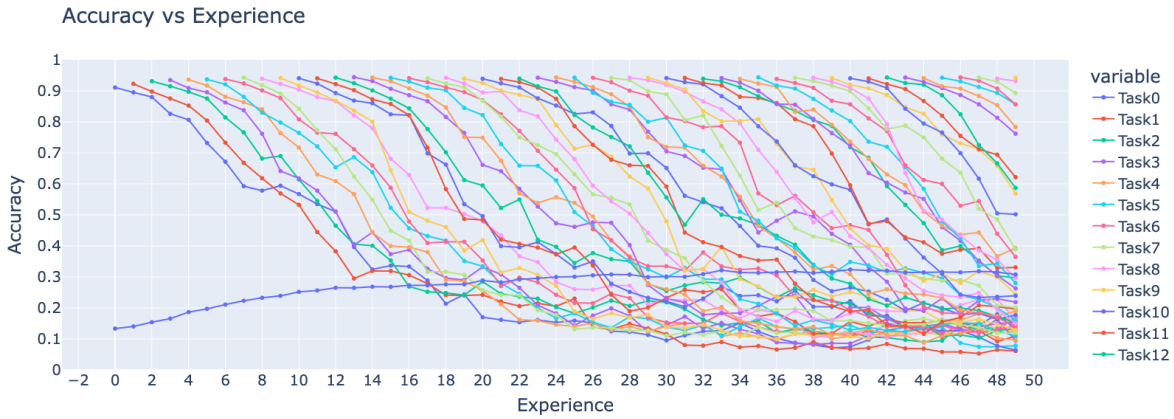


Figura 53: Accuracy en Naive con 50 tareas

La tabla 4 posee las métricas consideradas en este experimento. Esta tabla resalta más los resultados resumidos en el experimento anterior visibles en la tabla 3.

4.4.5. Comparación con saturación de experiencias

En este apartado se comparan las estrategias manteniendo la arquitectura de la red y la cantidad de tareas fijas en 10 pero reentrenando el modelo con las experiencias nuevamente. El objetivo es contrastar el desempeño de estrategias cuando se entrena nuevamente con experiencias vistas, cuantificando la mejora del desempeño al reaprender.

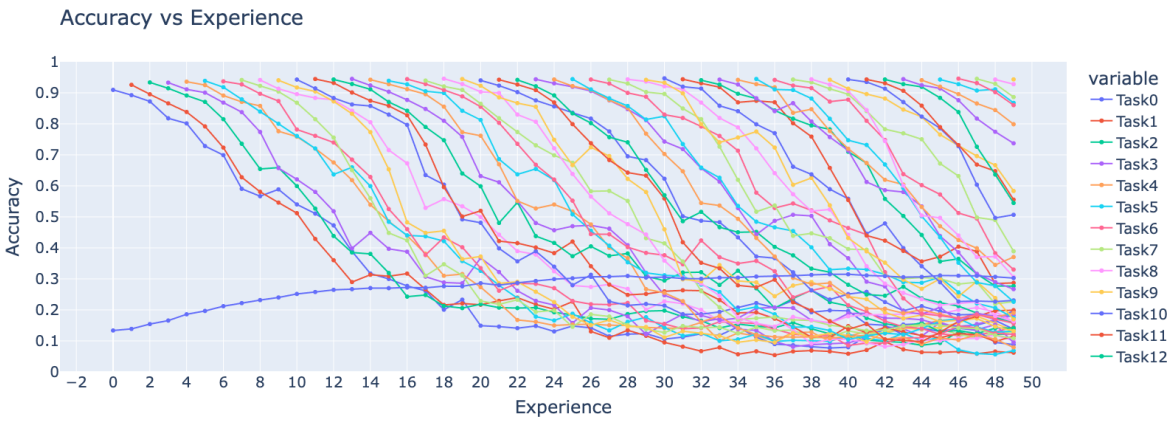


Figura 54: Accuracy en Propuesta 1 con 50 tareas

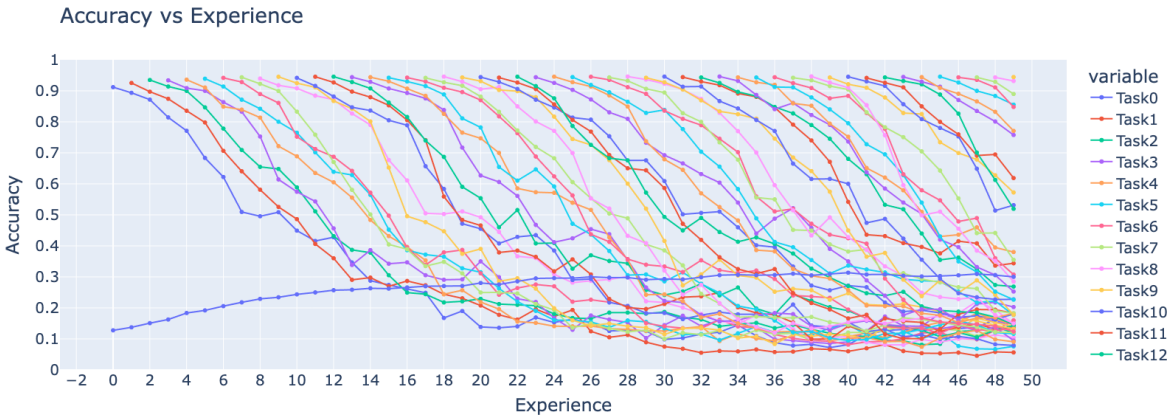


Figura 55: Accuracy en Propuesta 2 con 50 tareas

Métrica	Accuracy	Average Incremental Accuracy	Forgetting medio
EWC	0,357	0,560	0,465
Naive	0,308	0,543	0,521
Propuesta 1	0,303	0,538	0,528
Propuesta 2	0,300	0,528	0,537
Propuesta 3	**0,474**	**0,625**	**0,278**

Tabla 4: Métricas finales para entrenamiento de 50 tareas

Se filtra la cantidad de estrategias a comparar por las de mayor interés en este trabajo. El criterio utilizado fue escoger algoritmos que mostraran mejor desempeño en los casos estudiados en secciones anteriores. En concreto *EWC*, cuyo grado de olvido para las primeras tareas es destacable, *Naive* por ser un caso base y las estrategias propuestas por ser un elemento relevante de este trabajo, en especial la propuesta #3.

La arquitectura utilizada en este escenario se define en la Figura 5, es decir, la misma utilizada

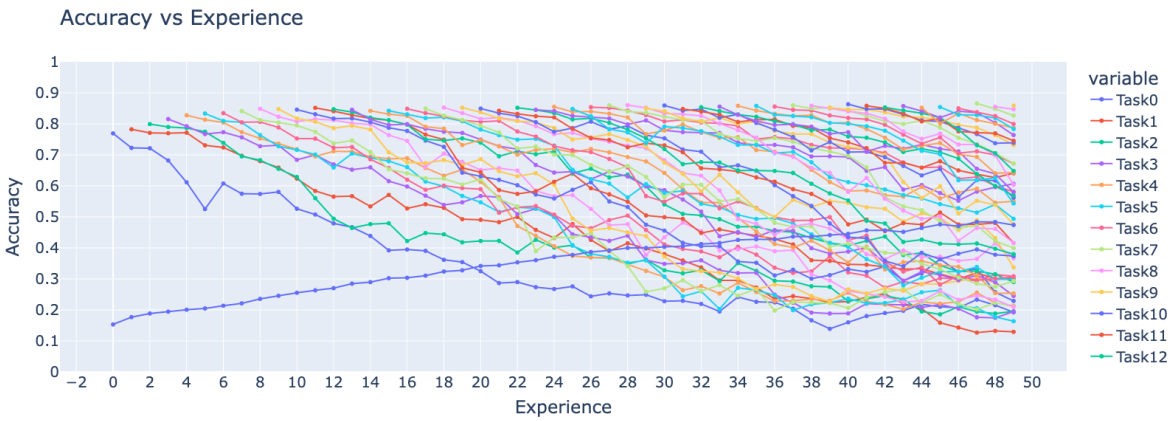


Figura 56: Accuracy en Propuesta 3 con 50 tareas

durante todos los experimentos anteriores.

En estos experimentos se entrenan a los modelos con 10 tareas en un total de 50 experiencias, es decir, se entrena con todos los datos un total de 5 veces.

En la Figura 57, se aprecia una debilidad grande de *Elastic Weight Consolidation* (al menos con los hiperparámetros actuales) respecto a la una estrategia ingenua y la propuesta 3, y es que la primera le cuesta consolidar el conocimiento de todas las clases a pesar de el constante reentrenamiento. No es capaz del todo de estabilizar su conocimiento en todas las tareas manteniéndose en un ciclo de olvido perpetuo. El gráfico muestra que en cada ciclo de reentrenamiento olvida menos, no obstante implicaría mucho tiempo hasta lograr matener su exactitud invariante.

La tarea que es de mayor dificultad aprender es la tarea 8, pues el modelo presenta decaimientos en su capacidad para clasificar correctamente las clases pertenecientes a esta tarea

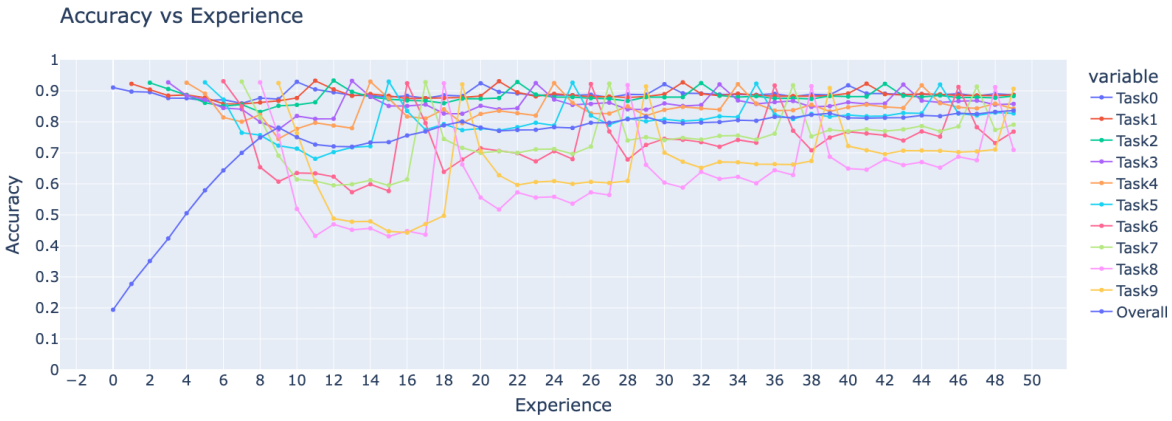


Figura 57: Reentramiento de experiencias en EWC

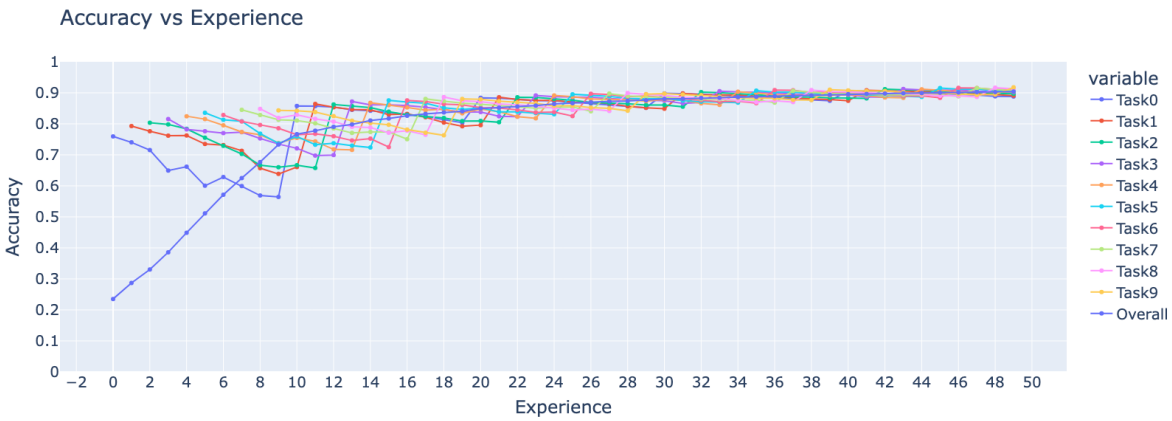


Figura 58: Reentramiento de experiencias en Naive

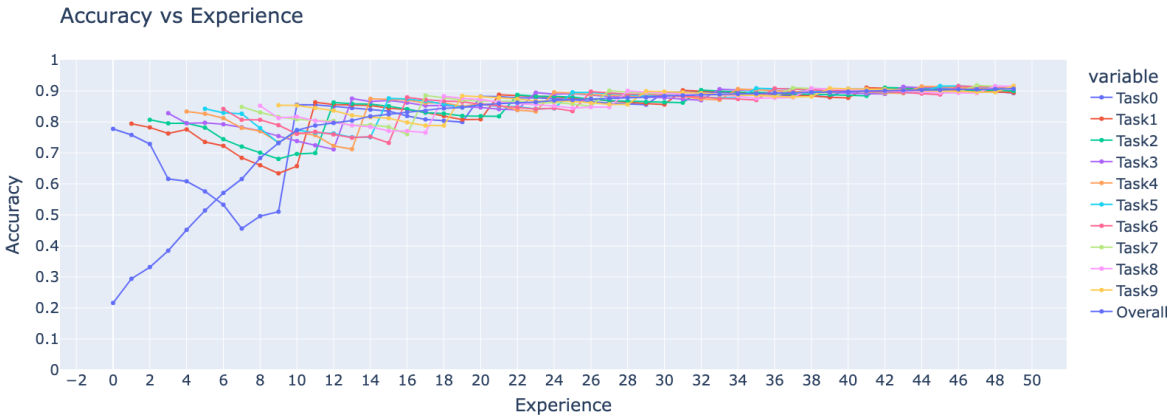


Figura 59: Reentramiento de experiencias en Propuesta 3

Métrica	Accuracy	Forgetting medio
EWC	0,836020	0,140080
Naive	0,903870	0,029109
Propuesta 3	**0,906470**	**0,027804**

Tabla 5: Métricas finales para reentrenamiento

Las estrategias ingenua y propuesta 3 se comportan muy similar, donde la propuesta 3 tiene mejor desempeño que la estrategia ingenua, sin embargo, esta diferencia es realmente mínima. La tabla 5 muestra las métricas con mayor precisión.

CAPÍTULO 5
CONCLUSIONES

5.1. Conclusiones del trabajo

A lo largo de este trabajo, se abordaron algoritmos realmente interesantes en la minimización del indeseado efecto de olvido catastrófico. Cada uno de estos algoritmos tenia sus ventajas frente a ciertos escenarios, no todos abordados durante la investigación, sin embargo obtener esas ventajas tenía consecuencias que se deben asumir. Lamentablemente no se pudieron comparar una gran variedad de otros algoritmos presentes en esta área del aprendizaje automático, cuyos resultados podrían haber inspirado nuevas propuestas, aportando aún más en el aprendizaje continuo.

Gran parte de las propuestas presentadas no tuvieron el éxito que se hubiese esperado, no obstante, así es la investigación, en donde no todo lo que se desarrolla debe necesariamente tener resultados sobresalientes respecto a lo presente en el estado del arte. Por otro lado, la última propuesta, optimización de uso de pesos, tuvo un éxito en escenarios importantes abordados en esta investigación.

Esto nos lleva a afirmar algo obvio en la adopción de estrategias, que es saber si lo escogido tendrá éxito. La respuesta es *depende*, depende del escenario, el contexto en donde se ve inmerso la estrategia seleccionada. En el contexto de este trabajo, la estrategia propuesta 3 funcionó mejor que *Elastic Weight Consolidation* frente a varias tareas o frente al reentreno con experiencias vistas, sin embargo con la consecuencia de olvidar las primeras tareas a medida que se entrenan nuevas tareas.

Las limitaciones principales de este trabajo son relacionadas principalmente con el poder de cómputo y el tiempo de espera necesario para llevar a cabo ciertas experiencias. En versiones preliminares de la investigación se intentó realizar *benchmarks* con los algoritmos *iCaRL* y *Orthogonal Gradient Descent*, sin embargo el uso de memoria en disco y/o tiempo de entrenamiento hicieron intratable continuar con estos algoritmos. Otras limitaciones se deben al pobre maneja de uso de *GPU* con el *framework* utilizado, lo que impidió obtener resultados en un tiempo razonable.

Los resultados presentados a través de gráficos y tablas no deben ser tomados como única referencia, sino que deben ser comprendidos en su contexto, pues los hiperparámetros usados para cada modelo son fruto de un proceso de *model selection* que puede diferir de otros trabajos del estado del arte. Trabajos como [Farajtabar *et al.*, 2020], [Wang *et al.*,], [De Lange *et al.*, 2021] y [Wang *et al.*, 2023] son otras referencias con resultados levemente diferentes interesantes de comprender, *¡no hay que olvidarlas!*

5.2. Trabajo futuro

Como ya se mencionó, en esta investigación se evaluó con algunos algoritmos que no fueron incorporados en la versión final, por lo que en un trabajo futuro es deseable que entren a ser parte de las comparaciones.

Durante el trabajo, se probaron más propuestas que quedaron muy cerca de ser incluídos en el trabajo. Uno de ellso era utilizar modelos de *Ensembled Learning* en combinación con estrategias de *Continual Learning*. Se tiene una especial confianza en combinar la estabilidad de *Elastic Weight Consolidation* de las primeras tareas con la gran capacidad de la propuesta #3 de retener nueva información. En esa misma línea, se probaron modelos ensamblados compuestos por los modelos resultantes de cada tarea entrenada, en donde, considerando que un modelo t habia visto la tarea 1 hasta la t, tenían decisión de voto (con un grado de confianza) solo si la tarea predicha pertenecía a alguna que se haya visto durante su fase de entrenamiento. Esta idea resulta interesante de seguir explorando en un futuro.

ANEXOS

Definiciones estadísticas

Verosimilitud

La función de verosimilitud o simplemente verosimilitud [Fisher, 1922] es una función que mide la probabilidad conjunta de que un conjunto de datos sean observados dado un parámetro asociado a la función de probabilidad de esos datos.

Formalmente, sea X un conjunto de datos observados, $f(X, \theta)$ la función de probabilidad de X y θ el parámetro asociado a la función de probabilidad f, entonces la verosimilitud ℓ se define como:

$$\ell(\theta, X) = f(X, \Theta = \theta) \tag{19}$$

Esperanza / Valor esperado de una variable aleatoria

Sea X una variable aleatoria con distribución de probabilidad $f(X)$, entonces su esperanza [Papoulis y Pillai, 2002] se define como

$$\mathbb{E}[X] = \int_{-\infty}^{\infty} (X = x) \cdot f(X = x) dx$$

El valor esperado de una transformación g sobre X es

$$\mathbb{E}[g(X)] = \int_{-\infty}^{\infty} g(X = x) \cdot f(X = x) dx$$

Varianza

Sea X una variable aleatoria, entonces su varianza [Papoulis y Pillai, 2002] se define como

$$Var(X) = \mathbb{E}[X^2] - \mathbb{E}[X]^2$$

Información de Fisher

La información de *Fisher* [Ly *et al.*, 2017] provee una forma de medir la cantidad de información que una variable aleatoria X contiene sobre un parámetro θ de la distribución de probabilidad asumida para esa variable aleatoria

Definición

Dada una variable aleatoria X que se asume que sigue una distribución de probabilidad $f(X, \theta)$, donde θ es el parámetro de la distribución, la información de *Fisher* se calcula como la varianza de la derivada parcial con respecto a θ de la log-verosimilitud $\ell(\theta, X)$

$$\mathcal{I}(\theta) = Var\left(\frac{\partial}{\partial\theta}\ell(\theta, X)\right) \tag{20}$$

Al aplicar la definición de varianza sobre la ecuación (20), se obtiene

$$Var\left(\frac{\partial}{\partial\theta}\ell(\theta, X)\right) = \mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\ell(\theta, X)\right)^2\right] - \mathbb{E}\left[\frac{\partial}{\partial\theta}\ell(\theta, X)\right]^2 \tag{21}$$

Desarrollando el primer término de la ecuación (21) se obtiene lo siguiente:

$$\mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\ell(\theta, X)\right)^2\right] = \int_{-\infty}^{\infty}\left(\frac{\partial}{\partial\theta}\ell(\theta, X = x)\right)^2 f(X = x, \theta)dx \tag{22}$$

Desarrollando el segundo término de la ecuación (21) se obtiene lo siguiente:

$$\begin{aligned}
\mathbb{E}\left[\frac{\partial}{\partial\theta}\ell(\theta, X)\right]^2 &= (\int_{-\infty}^{\infty}\frac{\partial}{\partial\theta}\ell(\theta, X = x)\cdot f(X = x, \theta)dx)^2 && (23)\\
&= \left(\int_{-\infty}^{\infty}\frac{1}{f(X = x, \theta)}\cdot\frac{\partial}{\partial\theta}f(X = x, \theta)\cdot f(X = x, \theta)dx\right)^2 && (24)\\
&= \left(\int_{-\infty}^{\infty}\frac{\partial}{\partial\theta}f(X = x, \theta)dx\right)^2 && (25)\\
&= \left(\frac{\partial}{\partial\theta}\int_{-\infty}^{\infty}f(X = x, \theta)dx\right)^2 && (26)\\
&= \left(\frac{\partial}{\partial\theta}(1)\right)^2 && (27)\\
&= 0 && (28)
\end{aligned}$$

Juntando el resultado obtenido en las ecuaciones (22) y (23), el valor de la información de Fisher es:

$$\begin{aligned}
\mathcal{I}(\theta) &= \mathbb{E}\left[\left(\frac{\partial}{\partial\theta}\ell(\theta, X)\right)^2\right] && (29)\\
&= \int_{-\infty}^{\infty}\left(\frac{\partial}{\partial\theta}\ell(\theta, X = x)\right)^2 f(X = x, \theta)dx && (30)
\end{aligned}$$

Matriz de información de Fisher

Sea X una variable aleatoria con distribución de probabilidad $f(X, \boldsymbol{\theta})$, donde $\boldsymbol{\theta}$ es un vector de parámetros asociados a la distribución, la matriz de información de *Fisher* (extensión de la información de *Fisher*) se define como:

$$\mathcal{I}(\boldsymbol{\theta})_{i,j} = \mathbb{E}\left[\left(\frac{\partial}{\partial\theta_i}\ell(\boldsymbol{\theta}, X)\right)\left(\frac{\partial}{\partial\theta_j}\ell(\boldsymbol{\theta}, X)\right)\right] \tag{31}$$

REFERENCIAS BIBLIOGRÁFICAS

[nag, 2021] (2021). Gradient descent with nesterov momentum from scratch. https://machinelearningmastery.com/gradient-descent-with-nesterov-momentum-from-scratch/. Accessed: 2024-01-17.

[mom, 2021] (2021). Momentum - cornell university. https://optimization.cbe.cornell.edu/index.php?title=Momentum. Accessed: 2024-01-17.

[Bar, 2024] (2024). Bard model. https://bard.google.com/. Accessed: 2024-01-14.

[Cha, 2024a] (2024a). Chain rule. https://mathworld.wolfram.com/ChainRule.html. Accessed: 2024-01-14.

[Cha, 2024b] (2024b). Chatgpt model. https://chat.openai.com/. Accessed: 2024-01-14.

[Lla, 2024] (2024). Llama model. https://ai.meta.com/llama/. Accessed: 2024-01-14.

[Amari, 1993] Amari, S.-i. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196.

[Bozinovski y Fulgosi, 1976] Bozinovski, S. y Fulgosi, A. (1976). The influence of pattern similarity and transfer learning upon training of a base perceptron b2. En *Proceedings of Symposium Informatica*, volumen 3, pp. 121–126.

[Burkov, 2019] Burkov, A. (2019). *The Hundred-Page Machine Learning Book*, pp. 3–4. Andriy Burkov.

[Bushaev, 2018] Bushaev, V. (2018). Understanding rmsprop-faster neural network learning. *Towards Data Science*.

[Chaudhry *et al.*, 2018] Chaudhry, A., Ranzato, M., Rohrbach, M., y Elhoseiny, M. (2018). Efficient lifelong learning with a-gem. *arXiv preprint arXiv:1812.00420*.

[Chong y Żak, 2013] Chong, E. y Żak, S. (2013). *An Introduction to Optimization*, pp. 131–160. Wiley Series in Discrete Mathematics and Optimization. Wiley.

[Crawshaw, 2020] Crawshaw, M. (2020). Multi-task learning with deep neural networks: A survey. *arXiv preprint arXiv:2009.09796*.

[De Lange *et al.*, 2021] De Lange, M., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., y Tuytelaars, T. (2021). A continual learning survey: Defying forgetting in classification tasks. *IEEE transactions on pattern analysis and machine intelligence*, 44(7):3366–3385.

[Díaz-Rodríguez *et al.*, 2018] Díaz-Rodríguez, N., Lomonaco, V., Filliat, D., y Maltoni, D. (2018). Don't forget, there is more than forgetting: new metrics for continual learning. *arXiv preprint arXiv:1810.13166*.

[Duchi *et al.*, 2011] Duchi, J., Hazan, E., y Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(7).

[Farajtabar *et al.*, 2020] Farajtabar, M., Azizan, N., Mott, A., y Li, A. (2020). Orthogonal gradient descent for continual learning. En *International Conference on Artificial Intelligence and Statistics*, pp. 3762–3773. PMLR.

[Farquhar y Gal, 2018] Farquhar, S. y Gal, Y. (2018). Towards robust evaluations of continual learning. *arXiv preprint arXiv:1805.09733*.

[Fisher, 1922] Fisher, R. (1922). *On the Mathematical Foundations of Theoretical Statistics*. Philosophical transactions of the Royal Society of London: Mathematical and physical sciences. Harison.

[French, 1991] French, R. M. (1991). Using semi-distributed representations to overcome catastrophic forgetting in connectionist networks. En *Proceedings of the 13th annual cognitive science society conference*, volumen 1, pp. 173–178.

[Gutstein y Stump, 2015] Gutstein, S. y Stump, E. (2015). Reduction of catastrophic forgetting with transfer learning and ternary output codes. En *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE.

[He y Jaeger, 2018] He, X. y Jaeger, H. (2018). Overcoming catastrophic interference using conceptor-aided backpropagation. En *International Conference on Learning Representations*.

[Hsu *et al.*, 2018] Hsu, Y.-C., Liu, Y.-C., Ramasamy, A., y Kira, Z. (2018). Re-evaluating continual learning scenarios: A categorization and case for strong baselines. *arXiv preprint arXiv:1810.12488*.

[Kingma y Ba, 2014] Kingma, D. P. y Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kortge, 2022] Kortge, C. A. (2022). Episodic memory in connectionist networks. En *12th Annual Conference. CSS Pod*, pp. 764–771. Psychology Press.

[Lewandowsky y Li, 1995] Lewandowsky, S. y Li, S.-C. (1995). 10 - catastrophic interference in neural networks: Causes, solutions, and data. En Dempster, F. N., Brainerd, C. J., y Brainerd, C. J., editores, *Interference and Inhibition in Cognition*, pp. 329–361. Academic Press, San Diego.

[Li y Hoiem, 2017] Li, Z. y Hoiem, D. (2017). Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947.

[Lomonaco *et al.*, 2021] Lomonaco, V., Pellegrini, L., Cossu, A., Carta, A., Graffieti, G., Hayes, T. L., Lange, M. D., Masana, M., Pomponi, J., van de Ven, G., Mundt, M., She, Q., Cooper, K., Forest, J., Belouadah, E., Calderara, S., Parisi, G. I., Cuzzolin, F., Tolias, A., Scardapane, S., Antiga, L., Amhad, S., Popescu, A., Kanan, C., van de Weijer, J., Tuytelaars, T., Bacciu, D., y Maltoni, D. (2021). Avalanche: an end-to-end library for continual learning. En *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2nd Continual Learning in Computer Vision Workshop.

[Lopez-Paz y Ranzato, 2017] Lopez-Paz, D. y Ranzato, M. (2017). Gradient episodic memory for continual learning. *Advances in neural information processing systems*, 30.

[Ly *et al.*, 2017] Ly, A., Marsman, M., Verhagen, J., Grasman, R. P., y Wagenmakers, E.-J. (2017). A tutorial on fisher information. *Journal of Mathematical Psychology*, 80:40–55.

[Maclin y Opitz, 2011] Maclin, R. y Opitz, D. W. (2011). Popular ensemble methods: An empirical study. *CoRR*, abs/1106.0257.

[McCloskey y Cohen, 1989] McCloskey, M. y Cohen, N. J. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. En *Psychology of learning and motivation*, volumen 24, pp. 109–165. Elsevier.

[McRae y Hetherington, 1993] McRae, K. y Hetherington, P. A. (1993). Catastrophic interference is eliminated in pretrained networks. En *Proceedings of the 15h Annual Conference of the Cognitive Science Society*, volumen 1, p. 2.

[Mitchell, 1997] Mitchell, T. M. (1997). Machine learning.

[Oyanedel, 2024] Oyanedel, J. (2024). tt2. `https://github.com/joyanedel/tt2`.

[Papoulis y Pillai, 2002] Papoulis, A. y Pillai, S. (2002). *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill electrical and electronic engineering series. McGraw-Hill.

[Rahaf y Lucas, 2019] Rahaf, A. y Lucas, C. (2019). Online continual learning with maximally interfered retrieval. En *NIPS*.

[Ratcliff, 1990] Ratcliff, R. (1990). Connectionist models of recognition memory: constraints imposed by learning and forgetting functions. *Psychological review*, 97(2):285.

[Rebuffi *et al.*, 2017] Rebuffi, S.-A., Kolesnikov, A., Sperl, G., y Lampert, C. H. (2017). icarl: Incremental classifier and representation learning. En *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010.

[Saad, 1999] Saad, D. (1999). *On-Line Learning in Neural Networks*. Publications of the Newton Institute. Cambridge University Press.

[Settles, 2009] Settles, B. (2009). Active learning literature survey.

[Wang *et al.*,] Wang, L., Zhang, X., Su, H., y Zhu, J. A comprehensive survey of continual learning: Theory, method and application. arxiv 2023. *arXiv preprint arXiv:2302.00487*.

[Wang *et al.*, 2023] Wang, Z., Yang, E., Shen, L., y Huang, H. (2023). A comprehensive survey of forgetting in deep learning beyond continual learning. *arXiv preprint arXiv:2307.09218*.

[Zeiler, 2012] Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.