TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Estudio de algoritmos de entrenamiento en redes neuronales cuantificadas

Autor

Francisco José Aparicio Martos

Directores

Rocío Celeste Romero Zaliz

Juan Bautista Roldán Aranda

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, 8 de julio de 2022

Estudio de algoritmos de entrenamiento en redes neuronales cuantificadas

Francisco José Aparicio Martos

Palabras clave: computación neuromórfica, red neuronal, cuantificación, Backpropagation, HSIC, Feedback Alignment, Synthetic Gradients.

Resumen

Las redes neuronales son una técnica de aprendizaje automático ampliamente utilizada en la actualidad. Permiten crear soluciones a problemas muy complejos como la visión por computador. Conforme avanza la investigación, las redes son cada vez más complejas y necesitan de equipos de altas prestaciones para entrenarlas y ejecutarlas. Estos equipos consumen grandes cantidades de energía insostenibles a largo plazo, por lo que se necesita buscar alternativas más eficientes energéticamente. En respuesta a esta necesidad surgió el campo de la computación neuromórfica. La computación neuromórfica estudia la creación de dispositivos de procesamiento que imitan el comportamiento del cerebro humano. Los dispositivos neuromórficos tienen altas velocidades de procesamiento, alto grado de paralelismo y bajo consumo energético, aunque tienen un gran inconveniente: limitada capacidad de representación numérica. Ésto provoca que las redes neuronales empleadas en estos circuitos tengan que estar cuantificadas, siendo necesario estudiar cómo diseñarlas para que puedan alcanzar el rendimiento de las redes con precisión elevada. Con el objetivo de avanzar en la investigación de la computación neuromórfica, en este Trabajo Fin de Grado se ha estudiado cómo afecta la cuantificación a distintos algoritmos de entrenamiento de redes neuronales. Para estudiar el comportamiento de la cuantificación sobre estos algoritmos, se ha experimentado sobre redes neuronales totalmente conectadas de distintos tamaños variando parámetros como: el número de capas, profundidad de la red, función de cuantificación y número de bits empleados. Basándose en la experimentación, se ha valorado la viabilidad de emplear estos algoritmos en circuitos neuromórficos, qué elementos de la computación neuromórfica se necesita mejorar y qué elementos se tienen que seguir investigando en futuros estudios.

Study of training algorithms for quantized neural networks

Francisco José Aparicio Martos

Keywords: neuromorphic computing, neural net, quantification, Backpropagation, HSIC, Feedback Alignment, Synthetic Gradients

Abstract

Neural networks are a machine learning technique broadly used nowadays. They allow to create solutions to complex problems like computer vision. As investigation progress the neural networks are getting more complex and need high performance devices to train and execute them. These devices spend high amounts of energy which are not sustainable in the long term, therefore we need to find more energetically effective alternatives. Neuromorphic computing is a new invetigation field that appear due to this problem. Neuromorphic computing studies the creation of brain-inspired computing devices. These devices have high computing speed, high paralelization degree and low power consumption, although there is a big disadvantage: limited numeric representation. Due to this problem, we need to quantize the neuronal networks to execute them in the neuromorphic circuits, so we need to study how to train these new networks. In order to contribute to the investigation in the field of neuromorphic computing, it has been studied how quantification affect to a variety of neural network training algorithms. To study the effect of quantification over these algorithms, it has been made experiments with fully connected neural networks of different sizes. These experiments varied parameters like number of layers, the deep of the net, quantification function and the amount of bits. Based on the experiments, it has been studied the viability of using these algorithms in neuromorphic circuits, which elements of neuromorphic computing need to be improved and future lines of investigation.

Yo, **Francisco José Aparicio Martos**, alumno de la titulación ingeniería informática de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 42233897D, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Firmado por APARICIO MARTOS, FRANCISCO JOSE (FIRMA) el día 08/07/2022 con un certificado emitido por AC DNIE 004

Fdo: Francisco José Aparicio Martos

Granada a 8 de julio de 2022 .

D. **Rocío Celeste Romero Zaliz**, Profesor del Área de Computación y Sistemas Inteligentes del Departamento Ciencia de la Computación e Inteligencia artificial de la Universidad de Granada.

D. **Juan Bautista Roldán Aranda**, Profesor del Área de Computación y Sistemas Inteligentes del Departamento Electrónica y Tecnología de Computadores de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Estudio de algoritmos de entrenamiento en redes neuronales cuantificadas***, ha sido realizado bajo su supervisión por **Francisco José Aparicio Martos**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 8 de julio de 2022 .

Los directores:

ROMERO ZALIZ ROCIO CELESTE - 78152361V

Firmado digitalmente por ROMERO ZALIZ ROCIO CELESTE - 78152361V
Fecha: 2022.07.08 12:42:34 +02'00'

Rocío Celeste Romero Zaliz **Juan Bautista Roldán Aranda**

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Uno de los grandes retos de la humanidad es la gestión de las inmensas cantidades de datos generados por la digitalización de la información, los avances tecnológicos y el uso intensivo de redes sociales. Para abordar esa cantidad de datos es necesario usar técnicas informáticas, en particular, el aprendizaje automático o machine learning. En este campo los algoritmos que más éxito tienen son los conocidos como **redes neuronales**. Su propósito es el de procesar la información, "aprender" de ella y ofrecer un resultado que dependerá del problema abarcado: clasificación, regresión o una combinación de ambos.

Para poder resolver problemas que cada vez son más complejos, procesar las grandes cantidades de información y realizar ejecuciones en tiempo real, se necesita que el hardware avance y permita una mayor potencia, una mayor paralelización y un mayor rendimiento energético. Estas necesidades son difíciles de satisfacer debido a varios obstáculos como son: la ralentización de la ley de Moore [2]; el problema de la barrera de la memoria o memory wall problem [3], brecha de velocidad que hay entre memoria y procesamiento; y el alto consumo energético de los procesos de entrenamiento, muy contaminantes para el medio ambiente [4]. Debido a estos problemas un campo que está emergiendo y es cada día más importante es el de la **computación neuromórfica** [5].

1.1. Motivación: Ventajas e inconvenientes de la computación neuromórfica

La computación neuromórfica es un nuevo tipo de computación la cual no se basa en la clásica arquitectura Von Neumann en la que memoria y procesamiento se encuentran separados. Este tipo de ordenadores se inspiran en el funcionamiento de nuestro cerebro y se componen de neuronas y

sinapsis.

El aspecto clave de la computación neuromórfica es el IMC (in memory computing), es decir, que tanto procesamiento como almacenamiento se llevan a cabo en el mismo lugar, dando como resultado un descenso en el tiempo computacional. Además de resolver el problema del memory wall, tiene otras características las cuales hacen de la computación neuromórfica una muy buena opción para las redes neuronales.

1. **Alto nivel de paralelismo**: debido a que los ordenadores neuromórficos se componen de neuronas y éstas pueden trabajar simultáneamente.
2. **Escalabilidad inherente**: Los ordenadores neuromórficos se pueden escalar fácilmente añadiendo más chips que incrementan el número de neuronas y sinapsis para poder albergar redes neuronales de mayor amplitud y profundidad.
3. **Computación dirigida por eventos**: significa que las neuronas no consumen prácticamente energía hasta que los datos estén listos para ser procesados, lo que hace a los ordenadores neuromórficos altamente eficientes.
4. **Estocasticidad**: debido al ruido existente en los circuitos se pueden añadir aleatoriedad de manera sencilla.

Todas estas propiedades hacen que la computación neuromórfica sea una opción idílica para las redes neuronales, pero hoy en día tiene una restricción muy importante y es su escasa precisión numérica (1-3 bits). Como consecuencia los algoritmos y arquitecturas de redes neuronales no obtienen el mismo rendimiento que en arquitecturas Von Neumann.

Debido a la importancia que tiene la computación neuromórfica en el futuro de la inteligencia artificial, a la continua investigación que hay por mejorar el rendimiento de estos dispositivos y al importante papel que tienen las redes neuronales actualmente y a futuro en el campo del machine learning, mi Proyecto Fin de Grado va a consistir en estudiar como se comportan varios algoritmos de entrenamiento para redes neuronales en los circuitos neuromórficos.

1.2. Objetivos del proyecto

Introducida la temática y las motivaciones para el desarrollo, me dispongo a describir los objetivos que se deberán de cumplir para completar el proyecto.

El **objetivo general** del proyecto es realizar un estudio del comportamiento de los algoritmos de entrenamiento de redes neuronales en circuitos neuromórficos. Para alcanzarlo se tendrán que ir completando una serie de subobjetivos más específicos.

- **Subobjetivo 1**: Analizar y comprender en profundidad los algoritmos de entrenamiento de redes neuronales.
- **Subobjetivo 2**: Estudiar el comportamiento de las versiones cuantificadas de los algoritmos de entrenamiento de redes neuronales.

1.3. Estructura de la memoria

La memoria se compone de 6 capítulos. Este primer Capítulo 1 realiza una introducción a la temática del proyecto y expone los motivos de su realización. Tras esta breve introducción, se muestra el capítulo de Preliminares 2 en el que se realiza una revisión del estado del arte de las redes neuronales, la cuantificación de las mismas y sobre la computación neuromórfica, además se presentarán: los algoritmos seleccionados mostrando su pseudocódigo y análisis algorítmico; las funciones de cuantificación a usar; y una breve definición sobre los memristores. Establecidas las bases teóricas el siguiente Capítulo 3 presenta la planificación del proyecto, mostrando la metodología a seguir durante el desarrollo del proyecto, los recursos usados y el presupuesto necesario para su elaboración. Los siguientes capítulos tratan sobre la experimentación, en el Capítulo 4 se definen los experimentos a realizar, como se van a realizar los análisis, qué conjuntos de datos se van a usar durante la experimentación y que herramientas se van a emplear. En el Capítulo 5 se muestran como se han realizado las experimentaciones y los resultados juntos a sus valoraciones. La memoria concluye con el Capítulo 6 en el que se realiza una valoración sobre el trabajo realizado en el que se analizan los resultados obtenidos y se muestran las vías de trabajo futuro sobre el tema.

Capítulo 2

Preliminares

En el campo del machine learning los algoritmos más usados son las redes neuronales. Con este tipo de algoritmos los ordenadores son capaces de realizar predicciones, encontrar patrones y replicar acciones que un ser humano puede hacer, como por ejemplo: el reconocimiento y la clasificación de objetos. Para que las redes neuronales puedan hacer estas tareas es necesario realizar un entrenamiento con grandes conjuntos de datos. En el proceso de entrenamiento a la red neuronal (en el caso del aprendizaje supervisado) se le muestra ejemplos de los problemas que tiene que resolver junto a su solución y en función de si ha conseguido resolver o no el problema, la red se actualizará. Pero, ¿qué es una red neuronal?

2.1. Redes neuronales

Una red neuronal [6] es un sistema de cómputo que consiste en un grafo dirigido ponderado en el que las neuronas artificiales son los nodos del grafo y los pesos de la red son las aristas ponderadas que conectan la salida de las neuronas con las entradas. Estas neuronas están formadas por dendritas (las conexiones de entrada) y por los axones (las conexiones de salida). Dentro de cada una de estas neuronas se realiza una suma ponderada con los pesos de las dendritas, y tras aplicarle una función de activación, se transmite el resultado a las neuronas de la siguiente capa por los axones.

El objetivo de entrenar una red neuronal es conseguir ajustar los pesos de la red para conseguir resolver problemas, ya sean de regresión, clasificación o una combinación de ambos. Los métodos que se usan para enseñar a la redes neuronales se conocen como algoritmos de entrenamiento. Un buen algoritmo de entrenamiento tiene que ser capaz de poder mejorar el desempeño de la red neuronal haciendo uso de los datos de entrenamiento del problema.

Actualmente el algoritmo más usado es el backpropagation [7] debido a

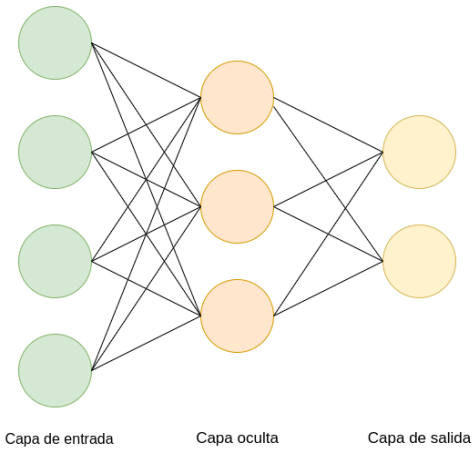


Figura 2.1: Red neuronal

su capacidad de encontrar buenos resultados y a su eficiencia. Desde que se inventó este algoritmo ha sido el más estudiado creándose numerosas variantes cuyo objetivo era el de mejorar los puntos débiles de este algoritmo, como son su inviabilidad biológica, el problema del transporte de pesos [8] y el desvanecimiento de gradientes.

Haciendo una revisión en la literatura actual nos podemos encontrar con algunos de los algoritmos más populares como son el grupo de los métodos quasi-newton, el método del gradiente conjugado[9] y el algoritmo de Levenberg-Marquardt[10].

2.1.1. Métodos Quasi-Newton

Los métodos quasi-newton son aproximaciones al método de Newton. La motivación de estos algoritmos es conseguir aplicar de manera práctica el método de Newton a problemas de machine learning actuales. El gran problema que tiene el método de Newton es su alto coste computacional, ya que hace uso de la derivada segunda para la optimización de funciones, es decir, necesita realizar el cálculo de la matriz Hessiana. En problemas con un número de variables grande es imposible aplicar este algoritmo. Por este motivo surgen los métodos quasi-newton, los cuales abordan el problema del cálculo de la matriz Hessiana mediante aproximaciones. Dentro del conjunto de estos algoritmos uno de los más conocidos es el L-BFGS [11] (Limited-Broyden–Fletcher–Goldfarb–Shanno), el cual aproxima la matriz Hessiana iterativamente creando un proceso mucho más eficiente.

2.1.2. Gradiente Conjugado

El gradiente conjugado es un algoritmo que se encuentra entre el gradiente descendente y el método de newton. La motivación de este algoritmo es solucionar la convergencia lenta del gradiente descendente y evitar los requerimientos del cálculo de la matriz Hessiana como invertir la matriz, evaluarla y almacenarla.

2.1.3. Método Levenberg-Marquardt

Finalmente, el método del Levenberg-Marquardt esta diseñado para trabajar con la función de pérdida. No necesita calcular la matriz Hessiana, sino que calcula el vector gradiente y la matriz Jacobiana.

Además de los métodos ya mencionados, existen alternativas que se centran en mejorar el propio backpropagation como puede ser el algoritmo Adadelta [12], el cual modifica el comportamiento del learning rate para mejorar la convergencia y los resultados que obtiene.

Vista la actualidad del entrenamiento de redes neuronales nos toca centrarnos en los otros dos aspectos que involucran este proyecto: la cuantificación de redes neuronales y la aplicación de redes neuronales en la computación neuromórfica.

2.2. Computación neuromórfica

En la actualidad la computación neuromórfica se encuentra en auge debido a los problemas anteriormente mencionados: ralentizacion de la ley de Moore, el memory wall problem y la contaminación producida por los procesos de entrenamiento de redes neuronales; y a sus grandes ventajas: alto nivel de paralelismo, escalabilidad inherente, computación dirigida por eventos y estocasticidad. Sin embargo, la computación neuromórfica se encuentra en una temprana edad y las numerosas investigaciones que se han realizado están lejos de la fase de producción [13]. Dentro de estos estudios los frameworks más destacados son: TrueNorth [14], SpiNNaker [15] y Loihi [16]. En el estudio de TrueNorth se han centrado en la aplicación de spiking neural networks [17], esta red es entrenada off-chip, es decir, no se entrenaba en el propio circuito y después se transformaba para ejecutarse en el circuito neuromórfico. SpiNNaker también aplica spiking neurons teniendo como ventaja principal una mayor flexibilidad en la elección de la topología de la red y el tipo de neurona a usar. Loihi al igual que los anteriores aplica las spiking neurons pero tiene como novedad el entrenamiento on-chip. Las otras redes

que se aplican en los circuitos neuromórficos son las fully connected y las convolutional neural networks, investigado como afecta la cuantificación a las redes una vez ya entrenadas. En este estudio [18] se ve como afecta la reducción de la precisión numérica de los pesos al rendimiento final de la red. Por otro lado, existen investigaciones que además estudian como afecta la cuantificación si se introduce en el entrenamiento [19]. Cabe destacar que para las redes fully connected solo se usan redes binarias mientras que la cuantificación de varios niveles solo se estudia en las redes convolucionales. [20]

2.2.1. Memristor

El memristor fue teóricamente formulado por Leon Chua en 1971 [21] y descubierto experimentalmente en los laboratorios de HP en 2008 [22]. Se trata de uno de los elementos más importante en la fabricación de circuitos neuromórficos debido a su capacidad de simular el comportamiento de las sinapsis cerebrales, lo que permite implementar redes neuronales directamente sobre el hardware. Los memristores son muy eficientes energéticamente, tienen una gran capacidad de escalabilidad y además son no volátiles, es decir, las sinapsis (los pesos de la red) que simule durante la ejecución pueden ser almacenadas. El aspecto clave de los circuitos neuromórficos que usan memristores es el In-Memory Computing (IMC), este término se refiere a que almacenamiento y cómputo se realizan en el mismo lugar, acelerando los cálculos realizados por la red. Para poder realizar el IMC, los memristores se disponen en una configuración especial conocida como matriz de barras transversales (crossbar array).

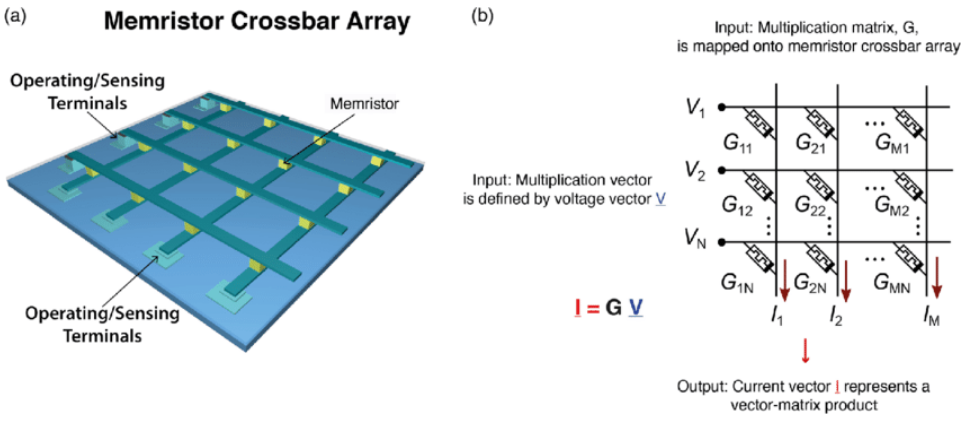


Figura 2.2: Crossbar Array [1]

El aspecto negativo de los memristores es su escasa precisión numérica, de 1 a 3 bits, lo que provoca que las redes que se ejecuten en este tipo de

circuitos tengan que estar cuantificadas, es decir, los pesos de la red tienen que representarse con enteros de 1 a 3 bits, representado entre 2 y 8 niveles de corriente.

2.3. Redes neuronales cuantificadas

La cuantificación de redes neuronales es un campo de estudio que se encuentra en auge debido a cuestiones como el tamaño que ocupan los modelos y las velocidades de inferencia [23]. Hoy en día los modelos más punteros consisten en arquitecturas de varios Gigabytes de tamaño, por lo que solo pueden ser ejecutadas en equipos que cuenten con la capacidad de almacenamiento y de cómputo necesarias para realizar las inferencias. Estas restricciones provoca que dispositivos móviles no puedan usar este tipo de redes de forma nativa. Otro gran problema que acarrea el uso de redes tan grandes, es que los equipos en los que se ejecutan necesitan grandes cantidades de energía, y por consecuencia, tienen un impacto ecológico muy negativo [4]. Por lo tanto, para reducir dicho consumo, se debe de ejecutar las redes en dispositivos que tengan menor consumo energético, como son los anteriormente mencionados, circuitos neuromórficos.

Con el objetivo de reducir los tamaños de los modelos, aumentar las velocidades de inferencia y reducir el consumo eléctrico se crean los métodos de cuantificación. Estos técnicas consisten en reducir la precisión numérica de la red, cuantificando los pesos y en ocasiones las operaciones dentro de la red. El proceso de cuantificar consiste en transformar los números de alta precisión en enteros de 1-8 bits, para ello existen varios métodos que se pueden dividir en procesos determinísticos y probabilísticos. Los procesos determinísticos mapean los números de alta precisión en el espacio de los números enteros de baja precisión, existiendo una relación de muchos a 1, un ejemplo es el método de redondeo que se usa en el BinaryConnect [24], donde se realiza una cuantificación binaria en función del signo. Por otro lado, los procesos probabilísticos usan distribuciones de probabilidad para realizar las cuantificaciones por lo que un número no siempre se cuantificará de la misma manera, ejemplo de este tipo de cuantificaciones se puede encontrar en [24] en el que la cuantificación estocástica se realiza en base a una distribución de probabilidad. La intuición es que si el valor es positivo existe una alta probabilidad de que su valor final sea 1, pero también cabe la posibilidad de que se cuantifique como -1.

Otro elemento importante de la cuantificación de redes neuronales es el momento en el que se aplican dichas cuantificaciones, pudiendo realizarse la cuantificación durante el entrenamiento o durante la inferencia. Durante el entrenamiento la cuantificación se aplica durante el feedforward pass y durante el backward pass mientras se realiza el entrenamiento. Implementa-

ciones de este tipo de cuantificaciones nos lo podemos encontrar en estudios como el ya mencionado BinaryConnect, donde se entrena una red binaria y en XNOR-Net [25] que aplica redes binarias a problemas más grandes como ImageNet. Por su parte la redes que usan cuantificación después del entrenamiento no aplican ninguna cuantificación a su entrenamiento, la red una vez entrenada es cuantificada para reducir su tamaño y acelerar la inferencia. Ejemplos de este tipo de cuantificaciones son: DeepCompression [26], reduce el tamaño de la red podando conexiones irrelevantes y los pesos restantes los cuantifica a valores discretos; Entropy-constrained scalar quantization [27], tiene en cuenta la información de la segunda derivada de la función de pérdida para medir la importancia de los distintos pesos, y la cuantificación se realiza a través de técnicas de clustering; e Incremental network quantization [28], cuyo proceso de cuantificación se divide en tres partes: partición de pesos, cuantificación por grupo y reentreno. [23]

2.4. Algoritmos a estudiar

Hasta ahora hemos analizando alternativas que no son viables computacionalmente para realizar computación neuromórfica ya que ni son alternativas como tal, si no más bien mejoras del propio backpropagation. A pesar de que estos ejemplos son los más ampliamente usados existen alternativas que no son tan usadas pero si suponen una alternativa real para el backpropagation. Estos son los algoritmos en los que se van a centrar el proyecto, pues el objetivo es estudiar como distintos algoritmos de entrenamiento se comportan bajo las condiciones de la cuantificación. Los algoritmos seleccionados para este estudio son: el HSIC Bottleneck [29], el feedback alignment [30] y synthetic gradients [31]. Se tratan de algoritmos recientes (2016-2020) los cuales han conseguido buenos resultados y gracias a su diseño pueden ser alternativas viables para la cuantificación. El algoritmo de HSIC ha sido seleccionado debido a que no necesita de una fase de retropropagación del error. El entrenamiento lo realiza en una única fase de propagación hacia delante, actualizando los pesos capa a capa. Por lo tanto, la ausencia de una propagación hacia detrás mejora sustancial el acarreo de error. La segundo alternativa, feedback alignment se ha seleccionado debido a la separación que hay entre los pesos usados en la propagación hacia delante y hacia atrás. Este algoritmo inicializa una matriz de pesos cuyo objetivo será propagar el error a los pesos reales de la red. Estos pesos se mantienen fijos por lo que solo se cuantifican al inicio, minimizando así el error de la cuantificación. Por último tenemos el algoritmo de synthetic gradients. El motivo por el que se ha seleccionado es su peculiar forma de calcular los gradientes, y es que este algoritmo usa redes neuronales auxiliares de menor tamaño para estimar los gradientes. De esta forma, no se propagan capa por capa el error,

conllevando un acarreo del ruido introducido por la cuantificación, sino que cada capa tiene su módulo auxiliar que le proporciona la estimación del gradiente. A continuación, se mostrará una descripción y análisis más detallado de cada algoritmo.

A continuación presento los algoritmos de entrenamiento que se van a estudiar para este proyecto.

2.4.1. Algoritmos preliminares

Este apartado esta destinado a explicar los métodos, algoritmos o herramientas que se van a utilizar en los algoritmos a estudiar con el objetivo de no tener que repetir su significado o funcionamiento.

Propagación hacia delante

La propagación forward o hacia delante es el proceso de inferencia de la red neuronal. Consiste en ir pasando la información de capa a capa, empezando por la inicial y llegando hasta la de salida. El proceso que se lleva a cabo es multiplicar la entrada por la matriz de pesos de la primera capa, a este resultado se le aplica la función de activación que se esté usando en la red. Este proceso se repite hasta llegar a la capa final, en la que la función de activación dependiendo del problema puede ser la identidad, softmax, entropía cruzada, etc.

Algorithm 1: Propagación hacia delante

Data: x: dato de entrada; $w = \{W^1, ..., W^L\}$: matrices de pesos;
L: número de capas; θ: función de activación
Result: h(x): hipótesis
$x^{(0)} \leftarrow x$
for *l = 1 to L* **do**
  $s^{(l)} \leftarrow (W^{(l)})^T x^{(l-1)}$
  $x^{(l)} \leftarrow \begin{bmatrix} 1 \\ \theta(s^{(l)}) \end{bmatrix}$
end
$h(x) = x^{(L)}$

Gradiente descendente

El gradiente descendente es una técnica básica de optimización de funciones en los algoritmos de machine learning. Consiste en calcular el mínimo

de una función de forma iterativa haciendo uso de la derivada de la función a estudiar.

La ejecución del algoritmo se inicia desde un punto aleatorio de la función. Para dirigirse hacia el mínimo más cercano el algoritmo calcula la derivada de la función en dicho punto y se desplaza en sentido contrario a esta.

Algorithm 2: Gradiente descendente

Data: w_0: punto de partida; θ: función a optimizar; max_iter: numero máximo de iteraciones; η learning rate
Result: un mínimo de la función
$w \leftarrow x_0$
for $i = 1$ *hasta* max_iter **do**
  $w = w - \eta f'(w)$
end

2.4.2. Backpropagation

El algoritmo Backpropagation, abreviatura de "backward propagation error", es un algoritmo de aprendizaje supervisado para redes neuronales que hace uso del gradiente descendente. Se trata de una generalización de la regla delta del perceptron para las redes neuronales multicapa. Su funcionamiento se basa en calcular el gradiente de la función de error con respecto a los pesos de la red. Para calcular los gradientes de cada capa primero se calculan los vectores de sensibilidad δ, los cuales cuantifican cuanto cambia el error con respecto a $s^{(l)}$ siendo $s^{(l)} = (W^{(L)})x^{(l-1)}$ el resultado de multiplicar los pesos de la capa l, $W^{(L)}$, con el resultado de la capa $l-1$, $x^{(l-1)}$. Calculadas las sensibilidades el siguiente paso es calcular los gradientes, para ello se multiplicara cada vector de sensibilidad, $\delta^{(l)}$, por los resultados de cada capa, $x^{(l)}$. Finalmente para actualizar la red se usará la regla del gradiente descendente.

Algorithm 3: Backpropagation para calcular la sensibilidad

Data: Un punto de la muestra (x, y); ; las matrices de pesos, $w = \{W^1, ..., W^L\}$; la función de activación θ
Result: La sensibilidad $\delta^{(l)}$ para $l = \{L, ..., 1\}$
Ejecutamos el proceso Feedforward 1 para obtener:
$s^{(l)}$ para $l = \{L, ..., 1\}$
$x^{(l)}$ para $l = \{L, ..., 1\}$
$\delta^{(L)} \leftarrow 2(x^{(L)-y})\theta'(s^{(L)})$
for $l = L - 1$ *hasta* 1 **do**
  $\delta^{(l)} = \theta'(s^{(l)}) \otimes [W^{(l+1)}\delta^{(l+1)}]$
end

Algorithm 4: Proceso del calculo de gradientes $g = \nabla E_{in}(w)$ y la función de error $E_{in}(w)$

Data: $w = \{W^1, ..., W^L\}$; $D = (x_1, y_1)...(x_N, y_N)$; x_i: dato de entrada iésimo; y_i: etiqueta asociada al dato iésimo; N: tamaño de la muestra
Result: el error $E_{in}(w)$ y los gradientes $g = \{G^1, ..., G^L\}$
Inicialización: $E_{in} = 0$ y $G^{(l)} = 0$ para $l = 1, ..., L$
for *por cada punto en la muestra* $(x_n, y_n), n = 1, ..., N$ **do**
  Calculamos $x^{(l)}$ para $l = 1, ..., L$ [forward propagation] Ver 1
  Calculamos $\delta^{(l)}$ para $l = 1, ..., L$ [backpropagation] Ver 3
  for $l = 1, ..., L$ **do**
    $G^{(l)}(x_n) = [x^{(l-1)}(\delta^{(l)})^T]$
    $G^{(l)} \leftarrow G^{(l)} + \frac{1}{N} G^{(l)}(x_n)$
  end
end

La actualización de los pesos se hará con el uso del gradiente descendente el cual dependiendo de la estrategia usada se hará tras calcular los gradientes con un solo punto de la muestra, con un subconjunto o con todos los datos.

Eficiencia

A continuación, en este subapartado voy a realizar el análisis de la complejidad computacional del algoritmo. Para poder realizar el análisis lo primero que tendremos que saber es cual es la complejidad computacional de la multiplicación de matrices, ya que es una operación que se usa en todas las fases del algoritmo.

Algorithm 5: Multiplicación de matrices

Data: A_{nxn} y B_{nxn}, matrices a multiplicar
Result: C_{nxn}, matriz producto
Inicialización: $C = 0$
for *i = 0 to n* **do**
  for *j = 0 to n* **do**
    for *k = 0 to n* **do**
      $C_{ij} = C_{ij} + A_{ik}B_{kj}$
    end
  end
end

La multiplicación de matrices se compone de 3 bucles, el primero recorre las filas, el segundo las columnas y el tercero se encarga de realizar la suma y producto de los componentes. Esta última operación, anidada en el tercer

bucle tiene complejidad $O(1)$. Al haber 3 bucles anidados y cada uno de estos se ejecutan n veces, esta operación se ejecutará n^3, por lo tanto la multiplicación de matrices tiene complejidad $O(n^3)$. Sabiendo esto podemos analizar la complejidad del backpropagation.

La primera fase del backpropagation es el feedforward, si recordamos este algoritmo lleva la entrada a traves de la red mediante multiplicaciones de matrices hasta conseguir la salida, es por ello que consta de L iteraciones, siendo L el número de capas. Por cada una de estas capas se realiza una multiplicación de matrices que es $O(n^3)$, por lo tanto la complejidad es $O(n^3L)$, como $n > L$, $O(n^4)$.

Realizado el feedforward se realiza el calculo del gradiente de la salida, cuya complejidad es $O(n)$, y se comienza con el calculo de los vectores de sensibilidad en un bucle que realiza $L - 1$ iteraciones. Para calcular cada vector se realiza una multiplicación de matrices de eficiencia $O(n^3)$, por lo tanto calcular estos vectores es $O(n^4)$. El siguiente paso es calcular los gradientes el cual tiene la misma eficiencia $O(n^4)$.

Los 3 pasos anteriores se realizaran n veces por lo tanto, $O(n(n^4 + n^4 + n^4))$, es decir, que el algoritmo de backpropagation tiene una complejidad computacional de $O(n^5)$.

2.4.3. HSIC Bottleneck

El algoritmo HSIC Bottleneck se introduce en el año 2019 como alternativa a backpropagation para el entrenamiento de redes neuronales. La propuesta se basa en la teoría de la información, más específicamente en los principios de la desigualdad de Fano y la información mutua [32]. En el contexto de las redes neuronales la desigualdad de Fano indica que la probabilidad de clasificar erróneamente depende de la entropía condicional $H(Y|X)$, siendo Y la etiqueta y X la entrada. Además la información mutua puede ser escrita de la siguiente forma: $I(X, Y) = H(Y) - H(Y|X)$. Debido a que la entropía de las etiquetas es constante con respecto a los pesos de la red, cuando la probabilidad de fallar en la clasificación es baja, $H(Y|X)$ es también bajo mientras que la información mutua es alta.

El algoritmo no usa directamente la información mutua debido a que se produciría un sobreajuste, es por ello que se emplea una aproximación del cuello de botella de información, llamado criterio de independencia HSIC. Este criterio sirve para caracterizar la dependencia entre las distintas capas. El objetivo es que por cada capa de la red, maximizar el HSIC entre la activación de la capa y la salida deseada, minimizando el HSIC entre la activación de la capa y la entrada.

Algorithm 6: Entrenamiento HSIC sin formato

Data: $X_j \epsilon R^{mxd_x}$: conjunto de datos de entrada j; $Y_j \epsilon R^{mxd_y}$: etiquetas asociadas a la entrada j; capa T_i parametrizado por $\{\theta | W_i b_i\}$; $i \epsilon \{1, ..., L\}$: iterador de capas; m : tamaño del batch; n : numero de muestras; α : learning rate; d_x: número de características; d_y: número de clases

Result: red HSIC entrenada $T_i(\cdot) : \mathbb{R}^{d_{i-1}} \rightarrow \mathbb{R}^{d_i}$, $i \epsilon \{1, ..., L\}$

for $j \epsilon \{1, ..., n/m\}$ **do**
  for $i \epsilon \{1, ..., L\}$ **do**
    $Z_i = T_{i-1}(Z_{i-1})$ $//Z_0 = X_j$
    $g_i = \nabla_\theta (nHSIC(Z_i, X_j) - \beta nHSIC(Z_i, Y_j))$
    $\theta_i \leftarrow \theta_i - \alpha g_i$
  end
end

Eficiencia

El algoritmo se compone de dos bucles, uno que recorre los minibatches y otro que recorre las capas, por lo tanto tenemos bucles de n/m iteraciones y L iteraciones, siendo n el número de muestras, m el tamaño de los minibatches y L el número de capas. Dentro de los bucles anidados encontramos una multiplicación de matrices cuya eficiencia es $O(n^3)$. La siguiente operacion es el calculo de HSIC que como se indica en el paper su complejidad computacional depende del número de datos que le pasemos m, siendo $O(m^2)$, como $m < n$, lo podemos escribir como $O(n^2)$. Finalmente el último paso se trata de una actualización de los pesos de la capa mediante gradiente descendente, por lo tanto tiene una eficiencia de $O(n^2)$.

Como ya sabemos las eficiencias de las operaciones que se realizan dentro de los bucles, solo nos falta multiplicarlas por los bucles para hallar la complejidad del algoritmo. Se trata de dos bucles anidados por lo tanto $O(n^2(n^3 + n^2 + n^2)) = O(n^5 + n^4 + n^4) = O(n^5)$.

2.4.4. Synthetic gradient

Synthetic Gradient es un algoritmo propuesto por la empresa DeepMind [1]. La característica principal de este algoritmo es que entrena la red mediante gradiente descendente haciendo uso de gradientes sintéticos, es decir, gradientes resultantes de la predicción de una red neuronal auxiliar. El problema que intenta resolver este algoritmo es el del bloqueo que se produce en el backpropagation. Cuando una capa envía información hacia delante, tiene

[1] `https://www.deepmind.com/`

que esperar a que esta información llegue hasta la capa de salida de la red, se calcule el error, se calcule el gradiente y que se propague capa por capa. Para eliminar esta espera se introducen entre las capas unos módulos llamados módulos de comunicación que se encargarán de calcular los gradientes sintéticos.

El funcionamiento que siguen estas redes es el siguiente: la red recibe la entrada y comienza con el proceso de feedforward. Suponiendo que introducimos un modulo de comunicación entre cada pareja de capas contiguas, en cada paso del feedforward la información se transmite a la capa i+1 y al modulo de comunicación i+1. Este modulo de comunicación (un perceptron multicapa MLP) procesará el mensaje y realizará una predicción del gradiente correspondiente. El gradiente se envía a la capa i y esta se actualizara por gradiente descendente. Cuando se alcanza la última capa se calcula el error y se calcula el gradiente real que se irá propagando hacia atrás. El gradiente real se usará para actualizar los módulos de comunicación.

Algorithm 7: Synthetic gradient (supongo un modulo por pareja de capas)

Data: $w = \{W^1, ..., W^L\}$; $D = (x_1, y_1)...(x_N, y_N)$
Result: Red entrenada por synthetic gradient
for *por cada punto en la muestra* $(x_n, y_n), n = 1, ..., N$ **do**
  for $i = 0$ *to* L **do**
    Feedforward a la capa $i + 1$ y al modulo $i + 1$
    Calculo de synthetic gradient (Feedforward red modulo)
    Actualizar capa i
  end
  Calcular gradiente ultima capa
  for $i = L - 1$ *to* 1 **do**
    Calcular gradiente con Backprop
    Actualizar modulo i
  end
end

Eficiencia

Con respecto a la eficiencia, estamos ante el algoritmo menos eficiente teóricamente, ya que además de realizar el backpropagation tiene que entrenar y ejecutar los módulos adicionales para el cálculo de gradientes sintéticos.

El primer paso de este algoritmo es el feedforward, (primer bucle interno). En este bucle se transmite la información hacia delante y se actualizará cada capa con el gradiente sintético calculado por los módulos correspondientes. Por cada capa se realiza una multiplicación de matrices para

propagar la información, $O(n^3)$, el calculo del gradiente por el modulo de comunicación, como se trata de un perceptron multicapa muy sencillo (1-3 capas) podemos asumir que la complejidad de este proceso es de $O(n^3)$, y por último se actualiza la capa gradiente descendente, $O(n)$. Esta primera fase del algoritmo tiene por tanto una eficiencia de $O(n(n^3 + n^3 + n)) = O(n^4)$.

El siguiente bucle del algoritmo se encarga de realizar el calculo del gradiente real y propagarlo hacia atrás para actualizar los módulos de actualización. El calculo del gradiente de cada capa se realiza en $O(n^3)$ y actualizar cada modulo tendrá un coste de $O(n^4)$, ya que consiste en actualizar mediante backpropagation un MLP. Este bucle se ejecuta otras n veces por lo que tendrá una eficiencia de $O(n(n^3 + n^4) = O(n^5)$.

Juntando ambos bucles, los cuales se ejecutarán n veces en función del número de muestras, obtenemos que la eficiencia final del algoritmo es de $O(n(n^4 + n^5)) = O(n^6)$.

Como apunte final cabe mencionar que se trata del algoritmo más costoso computacionalmente si este ejecuta de manera secuencial sin hacer uso de multithreading. La potencia de este algoritmo reside en que el entrenamiento de cada una de las capas se puede realizar de manera independiente haciendo uso de múltiples hilos lo que provocaría una gran disminución de los tiempos de ejecución. Para este trabajo no se va a realizar dicha paralelización ya que nuestro objetivo no es el de encontrar el algoritmo más eficiente. Para obtener más información sobre la comparación de rendimiento real entre backpropagation y synthetic gradients consultar [33].

2.4.5. Feedback alignment

El algoritmo de feedback alignment se trata de una alternativa a backpropagation que propone como mejora el uso de matrices aleatorias fijadas de antemano para la retropropagación de valores, en vez de usar la matriz traspuesta de los pesos de la red. Esta idea surge debido a que el algoritmo de backpropagation no es posible biológicamente ya que el cerebro debería de tener las mismas conexiones para las fases hacia delante y hacia atrás, cosa que no ocurre. El hecho de usar las mismas matrices de pesos en ambas fases acarrea el problema del transporte de pesos [8], cuyo nombre se debe a que para poder actualizar los pesos correctamente se "transportan"hacia detrás al propagar el error.

El algoritmo a pesar de usar pesos aleatorios fijados consigue encontrar las soluciones debido a la siguiente propiedad: cualquier matriz B de pesos aleatorias es válida para el aprendizaje si cumple $e^T W B e > 0$, siendo W la matriz de pesos y e el error. Es decir, que la señal de enseñanza de la matiz Be no difiera en más de 90^o a la usada por el backpropagation $W^T e$, por lo tanto B empuja a la red prácticamente en la misma dirección en la que lo

haría W. Para acelerar el aprendizaje se tendría que estrechar la diferencia entre las direcciones de ambas matrices, pero esto no es necesario ya que W se actualiza en base a B, lo que produce el alineamiento.

Algorithm 8: Feedback alignment

Data: Un punto de la muestra (x, y); $w = \{W^1, ..., W^L\}$: las matrices de pesos; $b = \{B^1, ..., B^L\}$: la matriz de pesos aleatorios ; θ: la función de activación
Result: La sensibilidad $\delta^{(l)}$ para $l = \{L, ..., 1\}$
Ejecutamos el proceso Feedforward 1 para obtener:
$s^{(l)}$ para $l = \{L, ..., 1\}$
$x^{(l)}$ para $l = \{L, ..., 1\}$
$\delta^{(L)} \leftarrow 2(x^{(L)-y})\theta'(s^{(L)})$
for $l = L - 1$ *hasta* 1 **do**
  $\delta^{(l)} = \theta'(s^{(l)}) \otimes [B^{(l+1)}\delta^{(l+1)}]$
end

El resto de procesos se realiza de manera idéntica al algoritmo de backpropagation. 2.4.2.

Eficiencia

Al realizar las mismas operaciones que el algoritmo de Backpropagation tiene la misma eficiencia, es decir, $O(n^5)$

2.4.6. Comparación de eficiencias

Para cada uno de los algoritmos de entrenamiento hemos realizado su estudio de eficiencia, pero, ¿por qué?. La respuesta es sencilla. En este estudio se va a llevar a cabo un proceso de cuantificación, el cual disminuye la precisión numérica con el que se realizan los cálculos. Esta disminución provoca errores en los cálculos realizados. A mayor número de operaciones realizadas, mayor es el error arrastrado, es por ello que es de especial interés conocer cuantas operaciones necesita cada algoritmo. A menor número de operaciones, menor será el error.

Comparemos entonces los algoritmos estudiados. Todos los algoritmos cuentan con una eficiencia de $O(n^5)$, a excepción de los synthetic gradients, que es de $O(n^6)$. Por lo tanto, desde el punto de vista teórico, el algoritmo que más se verá afectado por la cuantificación es el de synthetic gradients. El resto de algoritmos se encuentran empatados desde este punto de vista. Para encontrar cual se verá más afectado por la cuantificación, se tendrá que analizar los resultados de la experimentación.

2.5. Funciones de cuantificación

Las funciones que se van a aplicar en este estudio son: Uniform-ASYMM y Uniform-SYMM [34]. Se han elegido estas funciones debido a que consiguen disminuir la precisión numérica de manera efectiva y además son una aproximación data-free, es decir, no necesitan de un conjunto de datos para calibrar los modelos.

2.5.1. Uniform-ASYMM

Mapea los pesos en coma flotante al rango de los enteros tomando como rango el máximo y el mínimo del conjunto a cuantificar.

$$x_q = \text{round}((x_f - min_{xf})\frac{2^n - 1}{max_{xf} - min_{xf}}) \tag{2.1}$$

Siendo x_f el tensor original y x_q el vector cuantificado.

2.5.2. Uniform-SYMM

Al igual que el anterior se mapea al rango cuantificado, esta vez tomando un rango simétrico entorno al 0 y tomando como extremos el máximo valor absoluto del tensor a cuantificar.

$$x_q = \text{round}(x_f\frac{2^{n-1} - 1}{max|x_f|}) \tag{2.2}$$

Capítulo 3

Planificación

En este capítulo se van a definir la lista de tareas que componen el proyecto, se definirá la metodología a seguir, los tramos en los que se dividirá el tiempo de trabajo, el presupuesto y el análisis de riesgo.

3.1. Tareas

Para poder alcanzar los objetivos definidos en la Introducción 1.2 debemos de definir las tareas que tendremos que realizar.

En primer lugar para poder realizar el estudio necesitaré de un repertorio de algoritmos de entrenamiento que estudiar y evaluar, es por ello que la primera tarea **T1** será buscar en la literatura los algoritmos alternativos a backpropagation, que será el algoritmo de comparación base. Tras esto, tendré que analizarlos teóricamente por lo que la segunda tarea **T2** será obtener la complejidad computacional de los algoritmos seleccionados.

Hecho el análisis teórico se realizará la comparación de sus rendimientos con precisión numérica completa y con la precisión numérica reducida, **T3**.

Finalmente con la implementación y los resultados de los experimentos realizaré una evaluación del desempeño de los algoritmos y valoraré si es posible a día de hoy el entrenamiento de redes neuronales en este tipo de circuitos, **T4**.

3.2. Metodología

Una vez definidas las tarea y los objetivos del proyecto se debe de elegir la metodología de desarrollo que se va a usar. El objetivo de seguir una metodología es el de crear una planificación para asegurar que el proyecto se

elabora correctamente y se siguen una serie de procedimientos que garanticen la calidad del producto o servicio que se va a producir.

Actualmente existen varias alternativas las cuales se pueden englobar en dos grandes categorías: tradicionales y ágiles.

Las metodologías tradicionales se caracterizan por definir desde un inicio todos los requisitos y ciclos de trabajo, siendo estos muy poco flexibles. Los ciclos se organizan de manera lineal por lo que para comenzar con el ciclo siguiente se deberá de haber completado el anterior. Este tipo de metodologías se deben de usar en proyectos cuyo objetivo sea muy claro y se pueda conseguir mediante unos procedimientos fijos no susceptibles a variaciones.

Las metodologías ágiles, a diferencia de las tradicionales, son flexibles ante los cambios. Son metodologías incrementales compuesta de una serie de ciclos cortos que van construyendo el producto final mediante pequeñas aportaciones. Gracias a esta característica son muy fáciles de modificar en caso de que aparezcan inconvenientes o surjan cambios en los objetivos que se establecieron en un inicio.

Al estar ante un trabajo de investigación cuyos objetivos principales se conocen desde un inicio, pero sus fases de desarrollo están sujetas a cambios debido a la naturaleza de las investigaciones, la familia de metodología que mejor se adapta a este trabajo es la de desarrollos ágiles. Dentro de esta categoría he seleccionado la metodología scrum debido a su popularidad y a los buenos resultados que se consiguen con su aplicación.

3.3. Scrum

Scrum [35] es un marco de trabajo ligero que ayuda a la gente, equipos y organizaciones a generar valor a través de soluciones adaptativas a problemas complejos. Se basa en el empirismo y en el lean thinking. El empirismo defiende que el conocimiento viene de la experiencia y en hacer decisiones en función de lo que se observa, mientras que el lean thinking es una filosofía que se centra en lo importante y no en los desperdicios, entendiendo por desperdicio aquellos procesos que consumen más recursos de los necesarios. Scrum usa una aproximación iterativa e incremental cuyo objetivo es el de optimizar la predecibilidad y el control de riesgos.

El desarrollo de proyectos con Scrum se basa en los sprints: ciclos de trabajos cortos con una duración de entre 2 y 4 semanas. En cada uno de estos sprints se trabaja para completar una funcionalidad del producto o servicio a desarrollar. Las funcionalidades son definidas en lo que se conoce como Product Backlog.

El Product Backlog es una lista con los requisitos de nuestro proyecto

ordenados por prioridad, estando los elementos más valiosos al principio de la lista. Tanto prioridades como requerimientos evolucionan a lo largo del ciclo de desarrollo y se someten a revisión durante las reuniones de análisis.

Dentro del marco scrum a las personas se les asigna un rol que tendrán que cumplir. Dichos roles son los siguientes:

- **Product Owner**: Encargado de maximizar el retorno de inversión (ROI) mediante la identificación de características de producto trasladándolas a características en la lista de prioridades, decidiendo cual debería estar en la cima del siguiente sprint. Además es el intermediario entre clientes y equipo de desarrollo y negocia con los stakeholders.

- **Equipo de desarrollo**: Son los encargados de desarrollar el producto que el cliente va a utilizar. Son equipos interdisciplinares y cuentan con capacidad de auto-gestión y planificación.

- **Scrum Master**: El ScrumMaster es el encargado de ayudar al equipo ha conseguir los objetivos asegurándose de que se aplica la metodología Scrum. No es un director al uso pues se encarga de enseñar al Product Owner y al equipo como aplicar correctamente Scrum y protege al equipo de impedimentos que ocasionen retrasos en las entrega.

Los sprints se componen de una serie de fases en las que se definen y analizan la tareas que se van realizar. Dichas fases son:

- **Planificación del sprint**: Esta reunión se compone de 2 fases, una primera en la que el Product Owner se reúne con el equipo y eligen del Product Backlog que tareas se deben de realizar. La segunda fase es una reunión del equipo de desarrollo en sí y se establece como se va a abordar las tareas seleccionadas.

- **Reunión diaria**: Consiste en una reunión de unos 15 minutos en la que el equipo se reúne para comentar qué trabajo ha elaborado, qué trabajo se tiene planeado realizar y problemas o impedimentos que hayan surgido.

- **Trabajo de desarrollo durante el sprint**: Se debe de realizar el trabajo que ha sido definido y en caso de que aparezcan muchos cambios se deberá de replanificar para evitar retrasos.

- **Revisión del Sprint**: Finalizado el sprint el equipo junto al Product Owner valoran el trabajo durante el mismo, detectando problemas y redefiniendo en caso de que sea necesario, elementos del Product Backlog.

- **Retrospectiva del Sprint**: Consiste en un análisis de qué ha salido bien y que cosas se pueden mejorar para hacer más eficientes los procesos.

3.4. Aplicación de Scrum al proyecto

A pesar de que el trabajo no consiste en la elaboración de un proyecto software como tal, las metodologías de trabajo y coordinación son imprescindibles en los trabajos de investigación, por lo que a continuación voy a describir como voy a aplicar esta filosofía a mi proyecto de investigación.

Los roles a asignar son: Scrum Master, Produc Owner y equipo de desarrollo. Mis roles serán los del equipo de desarrollo, ya que el proyecto va a ser realizado por mi mismo, por lo que implícitamente soy también el Scrum Master, pues durante el trabajo me asegurare de seguir la metodología definida. Por su parte, la Product Owner será mi tutora Rocío Romero Zaliz, ya que su trabajo consiste en guiarme a través del proyecto, estableciendo los objetivos a cumplir.

Con los roles ya definidos queda establecer la planificación que voy a seguir durante estos 3 próximos meses. En la segunda reunión (2-03-22), tras realizar un primer análisis sobre la temática del proyecto se definió conjuntamente las fases que debería de componer el TFG, y a partir de este esquema inicial se divide los 3 siguientes meses en los sprints a realizar.

3.4.1. Sprints

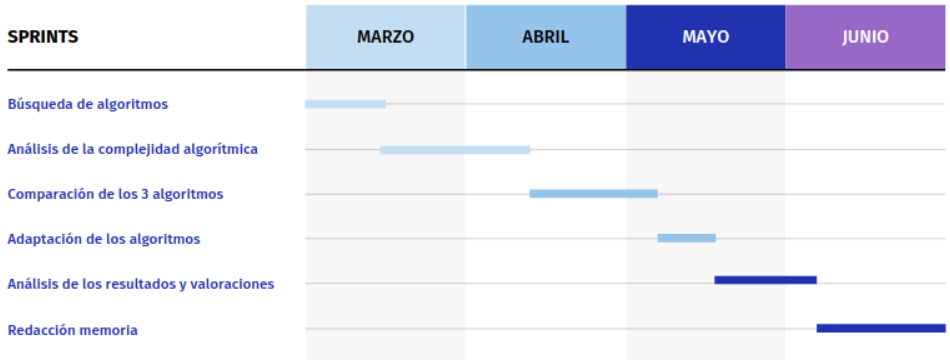
En el esquema inicial se estableció que las fases que componen este trabajo son las siguientes: búsqueda de algoritmos, análisis de la complejidad algorítmica, comparación de los 3 algoritmos, adaptación de los algoritmos, análisis de resultados y valoraciones, además paralelamente a todas estas fases hay que ir redactando en la memoria el trabajo realizado.

Tras haber realizado las estimaciones de el tiempo que puede durar cada una de las fases se han definido los siguientes sprints:

- **Sprint 1, Búsqueda de algoritmos**: Primera fase del proyecto en la que se buscará alternativas al backpropagation para poder realizar el estudio.

- **Sprint 2, Análisis de la complejidad algorítmica**: Seleccionado los algoritmos a estudiar es interesante conocer cuales son sus propiedades a niveles de eficiencia.

- **Sprint 3, Comparación de los 3 algoritmos**: En esta fase se implementan todos los algoritmos y se ejecutan para comparar sus resultados.
- **Sprint 4, Adaptación de los algoritmos**: Con una implementación funcional ahora se deberá de realizar las cuantificaciones pertinentes para poder realizar el estudio deseado.
- **Sprint 5, Análisis de resultados y valoraciones**: Se ejecutan los modelos sobre 1 o 2 bases de datos y se estudia como afecta la cuantización a su comportamiento, sacando finalmente las conclusiones y valoraciones del trabajo realizado.
- **Sprint 6, Redacción memoria**: Esta ultima fase consiste en terminar de redactar la memoria del proyecto, realizando todas la explicaciones teóricas y prácticas, mostrar resultados de experimentación, etc. Este sprint se realizará a lo largo de junio para terminar de plasmar el trabajo realizado.



3.5. Gestión de recursos

En este sección se presentan los recursos tanto humanos, hardware y software que se han empleado para la elaboración del proyecto.

3.5.1. Recursos humanos

- **Tutora del proyecto**: Dña. Rocío Celeste Romero Zaliz, profesora de la Universidad de Granada del departamento de Ciencias de la Computación e Inteligencia Artificial
- **Alumno:** Francisco José Aparicio Martos, alumno de Ingeniería Informática de la Escuela Técnica Superior de Ingenierías Informática

y Telecomunicaciones de la especialidad de Computación y Sistema Inteligentes.

3.5.2. Recursos hardware

Para la realización de este proyecto no se ha necesitado de recursos adicionales a los ya poseídos por el alumno. Todo el hardware ha sido aportado por el alumno siendo este:

- **Portátil**: MSI GE63 Raider con un procesador intel core i7 de séptima generación y una tarjeta gráfica geforce gtx 1050 ti. Se ha usado en las labores de programación del proyecto, ejecución de pruebas y redacción de memoria.
- **Pantallas**: Dos pantallas, ASUS y SAMSUNG, de 24 pulgadas cada una.

3.5.3. Recursos Software

Todo el software empleado durante este proyecto es de uso gratuito englobándose gran parte en la categoría de software libre. A continuación se lista todo el software usado en el proyecto:

- **Sistema Operativo**: Ubuntu Linux, versión 18.04.
- **Python**: más concretamente la versión 3.8.13, junto a python se usarán librerías incluidas en el propio python como por ejemplo: numpy y matplotlib, y librerías de uso libre que se encuentran subidas en la plataforma github. Más detalles sobre las librerías empleadas se puede encontrar en la sección de metodología.
- **Conda**: gestor de paquetes y de entornos para python.
- **Google Meet**: plataforma usada para realizar las reuniones por videollamada con la tutora.
- **Overleaf**: plataforma online para la redacción de documentos en LaTeX.
- **Git**: software para el control de versiones.
- **Github**: plataforma donde se alojará la memoria y el código del proyecto.
- **TimeCamp**: aplicación online para contabilizar cantidad de horas trabajadas.

3.6. Presupuesto

A continuación se presenta la estimación de los costes del proyecto teniendo en cuenta los recursos humanos, software y hardware presentados anteriormente.

3.6.1. Costes de recursos humanos

En este apartado tenemos que incluir el coste del trabajo que he realizado.

Para realizar una estimación de cuanto costaría el trabajo realizado tenemos que ver cuál es el sueldo medio de un recién graduado en ingeniería informática. Según el portal jobted [1] el sueldo de un recién graduado en ingeniería informática gira entorno a los 20400 euros al año, por lo que si lo dividimos entre 12 meses, 20 días laborales por mes aproximadamente y 8 horas de jornada diaria, da como resultado un sueldo de 10.63 €/h. Teniendo en cuenta que las horas empleadas en el proyecto han sido 300 horas. Obtenemos que el coste de mi trabajo es de 3189 euros.

3.6.2. Costes Software

Como he comentado en el apartado de los recursos, todo el software que se ha usado en la elaboración de este proyecto es de uso gratuito por lo tanto estos recursos no añaden ningún tipo de coste al proyecto.

3.6.3. Coste Hardware

Los recursos hardware usados han sido mi ordenador y las pantallas que se han mencionado en la sección de recursos. Para calcular su valor hay que tener en cuenta que los productos electrónicos se devalúan con el paso de los años, para realizar este cálculo haremos uso del método de depreciación lineal, el cual nos indica cuanto valor pierde nuestro dispositivo por año.

$$Depreciacion = \frac{valor\ inicial - valor\ residual}{vida\ util\ en\ años} \tag{3.1}$$

Siendo el valor residual:

$$Valor\ residual = \frac{valor\ inicial}{vida\ util\ en\ años} \tag{3.2}$$

Este valor indica el valor final que tendrá el producto una vez llegue al fin de su vida útil. Para el caso de los dispositivos electrónicos la vida útil

[1] `https://www.jobted.es/salario/ingeniero-inform%C3%A1tico`

según la agencia tributaria española [2] es de 10 años. Por lo tanto los valores actuales de cada uno de los dispositivos es de:

Recurso	Valor inicial	Valor residual	Antigüedad	Valor actual
Portátil	1200€	120€	4	768€
Monitor Samsung	190€	19€	2	155.8€
Monitor ASUS	120€	12€	4	76.8€

Cuadro 3.1: Tabla con los precios de los recursos hardware

3.6.4. Presupuesto total

Teniendo en cuenta todos los costes previamente calculados presento la siguiente tabla con el coste total del proyecto:

Concepto	Importe
Recursos Hardware	
Portátil	768€
Monitor Samsung	155.8€
Monitor ASUS	76.8€
Recursos Humanos	
Trabajo individual	3189€
	Total: 4189.6€

Cuadro 3.2: Presupuesto total del proyecto

3.7. Análisis de riesgos

Durante el desarrollo de cualquier proyecto siempre surgen imprevistos que alteran la programación inicial. Todos estos acontecimientos inesperados se conocen como riesgos. Para evitar que dichos riesgos afecten a la consecución de los objetivos, se deben de establecer una serie de protocolos a seguir

[2] `https://acortar.link/ytonfm`

para mitigar dichos problemas lo antes posible y conseguir que no afecten al trascurso del proyecto. Es por ello que antes de comenzar cualquier proyecto se debe de realizar una estimación acerca de los riesgos que pueden surgir y establecer planes de actuación. A continuación se presentan los riesgos que se han estudiado junto a sus causas y planes de actuación.

Código	Riesgo
1	No existe implementación de los algoritmos
2	No se encuentran alternativas viables
3	Las herramientas software no funcionan correctamente
4	Incompatibilidad de librerías
5	No hay feedback del tutor
6	Pérdidas de código o datos
7	Se estropea el ordenador de trabajo

Cuadro 3.3: Tabla con los riesgos del proyecto

Riesgo	Causa
1	Los autores no han hecho públicos sus códigos
2	No se han realizado estudios que consigan resultados buenos
3	Bugs propios de los producto software
4	Librerías que no se tenían pensadas para trabajar juntas
5	No ha podido asistir a la reunión por incompatibilidad horaria
6	Apagón repentino, fallo del almacenamiento
7	Sobrecalentamiento o golpe fortuito

Cuadro 3.4: Tabla con las causas de los riesgos

Riesgo	Plan de actuación
1	Implementar el algoritmo a mano o de no ser posible buscar otra alternativa
2	Probar con alternativas del propio backpropagation
3	Buscar alternativa o corregir el error a mano
4	Buscar otro conjunto de librerías compatibles
5	Planificar la reunión para otro día o mandar duda por correo
6	Recuperar datos a través del control de versiones
7	Comprar otro ordenador

Cuadro 3.5: Tabla con los planes de actuación para cada riesgo

Probabilidad / Impacto	0.1	0.3	0.5	0.7	0.9
Muy bajo		R5			
Bajo					
Medio		R3	R4		
Alto		R6			
Muy alto	R7	R2	R1		

Cuadro 3.6: Matriz probabilidad-impacto de riesgos

Capítulo 4

Diseño de experimentos e implementación

En este capítulo voy a describir los experimentos que voy a realizar en el estudio, las herramientas que usaré y cuales son mis hipótesis iniciales.

4.1. Experimentos a realizar

Para estudiar como afecta la cuantificación a los distintos algoritmos de entrenamiento presentados voy a realizar varios entrenamientos con distintos parámetros sacando métricas que me den información acerca de su desempeño y del comportamiento de sus pesos.

Los parámetros que voy a modificar en cada uno de los entrenamientos son los siguientes:

- **Profundidad de la red**: Cantidad de capas que conforman la red.
- **Anchura de las capas**: Número de neuronas en las capas ocultas.
- **Función de cuantificación**: Función usada en el proceso de cuantificación de los pesos.
- **Nivel de cuantificación**: Especifica si la cuantificación se realiza a nivel global, toda los pesos de la red estarán en un rango [min_global, max_global] fijado previamente antes de la ejecución; o local, los pesos de cada capa se cuantificarán en el rango [min(capa),max(capa)], este rango irá cambiando a lo largo del entrenamiento.
- **Número de bits**: Cantidad de bits que se usarán en el proceso de cuantificación.

Los parámetros de profundidad y anchura afectan a la arquitectura de la red mientras que la función de cuantificación, nivel de cuantificación y el número de bits modifican el proceso de cuantificación.

4.1.1. Arquitecturas

Las arquitecturas analizadas han sido seleccionadas partiendo del estudio [18]. Se ha seleccionado la arquitectura que mejor desempeño tenía por debajo de las 4096 sinapsis, número de sinapsis por chip, con la idea de evaluar una red que pueda ser ejecutada con la necesidad de un único crossbar array. Para estudiar como afecta profundidad y anchura se han seleccionado de manera arbitraria 5 arquitecturas extra para ver si estos parámetros permiten mejorar el desempeño de las redes cuantificadas. A continuación se muestran las arquitecturas seleccionadas:

Capas Ocultas	**Unidades ocultas**	**Sinapsis**
1	4	3190
6	50	18010
2	100	89610
3	100	99710
2	50	42310
3	50	44860

Cuadro 4.1: Arquitecturas FdNN

Todos estos modelos siguen el esquema fully connected neural network (redes totalmente conectadas), es decir, todas las neuronas están conectadas con todas las neuronas de la capa previa y de la capa posterior.

4.1.2. Conjuntos de datos

Las bases de datos que han sido seleccionadas para realizar el benchmark son MNIST [36] y Fashion-MNIST [37]. Ambas bases de datos son usadas para evaluar el rendimiento de algoritmos de aprendizaje automático. A día de hoy ya se han conseguido resolver casi a la perfección pero aun así se siguen usando para evaluar modelos que se encuentran en sus primeras fases de estudio, como es nuestro caso con los modelos cuantificados.

MNIST es una base de datos de números escritos a mano, estos números

se representan por imágenes de 28x28 en escala de grises, contando con un total de 70000 imágenes etiquetadas entre 0-9. Se dividen en un cojunto de entrenamiento de 60000 imágenes y un conjunto de test de 10000 imágenes.

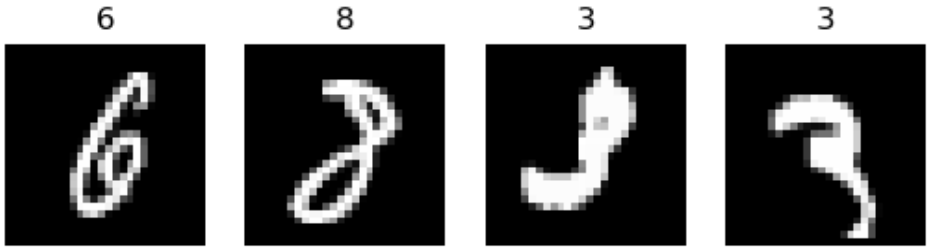


Figura 4.1: MNIST

Fashion MNIST es una base de datos de prendas de ropa (Camisetas/-tops, pantalones, suéteres, vestidos, abrigos, sandalias, camisas, zapatillas, bolsos y botas) con las mismas características que MNIST, un total de 70000 imágenes en escala de grises de 28x28, 60000 destinadas a entrenamiento y 10000 destinadas a test. A pesar de tener un mismo tamaño se trata de una base de datos más compleja de aprender y que representa mejor los problemas reales.

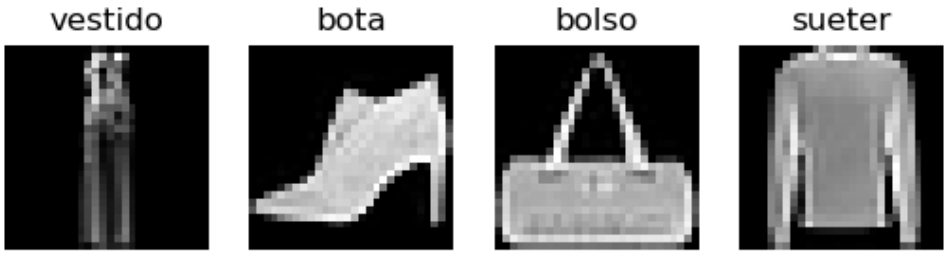


Figura 4.2: FMNIST

Al tener ambas las mismas características no hace falta que modifiquemos las estructuras de las redes, pues la entrada, imágenes de 28x28, y la salida, clasificación entre 10 clases, son las mismas.

4.1.3. Método de cuantificación

Ya tenemos las redes que vamos a usar en la experimentación y sobre que conjunto de datos las vamos a ejecutar. Ahora toca explicar como voy a simular de manera aproximada el comportamiento de las redes neuronales

en circuitos neuromórficos. Como se ha mencionado en los preliminares 2 los memristores cuentan con una precisión numérica limitada. Por lo tanto, los pesos se tienen que codificar con en esta precisión numérica. Las funciones de activación, las entradas y las salidas no es necesaria cuantificarlas debido al carácter analógico de los circuitos neuromórficos. Teniendo esto en cuenta, habrá que modificar el proceso de entrenamiento para que cada vez que se actualicen los pesos, cuantificarlos dentro de los rango definidos. Según hemos establecido los parámetros, se pueden diferenciar dos tipos de cuantificación, una a nivel global, mismo rango para toda la red, y nivel local, un rango distinto por cada capa. El nivel global es bastante simple, se define un rango inicial que tendrá x valores posibles (en función del número de bits) y los pesos solo podrán tomar esos valores. Es la simulación más fiel a la realidad debido a que los memristores se suelen fabricar con los mismos niveles (rango de valores). Mientras tanto, la cuantificación local consiste en cada vez que se actualice la red, se cuantifica capa por capa, cogiendo los máximos y mínimos de cada capa y cuantificando entre esos dos valores. Esto provoca que el intervalo sea dinámico durante la ejecución, cosa que no ocurre en la realidad. Esta cuantificación se ha explorado con la idea de estudiar que ocurriría si cada capa tuviera un rango de valores distintos, y si implica una mejora notable como para estudiar en un futuro si tener rangos distintos por capas es rentable, y de ser así, estudiar como seleccionar dichos rangos.

La cuantificación también se verá afectada por la función que usemos para cuantificar ASYMM 2.1 o SYMM 2.1, en el caso de la función ASYMM el rango estará comprendido entre el máximo y mínimo de la capa, mientras que con la función SYMM el intervalo es simétrico con respecto al 0 cuyos extremos son [-maximo absoluto, maximo absoluto].

4.1.4. Métricas y función de error

Definida la fase experimental ahora queda describir cómo se van a evaluar los resultados. Para ello tendremos que elegir bien las métricas que usaremos para valorar el desempeño de las redes entrenadas y cuál será la función de error que usaremos para ajustar los modelos.

Ambas bases de datos se engloban dentro de los problemas de clasificación, es decir, dado una muestra tendremos que establecer a qué métrica pertenece. Por lo tanto, métricas y función de error tienen que pertenecer a la clase de los problemas de clasificación.

Función de error

Para seleccionar la función de error tenemos que tener en cuenta el tipo de problema y la función de salida de nuestra red. Al estar ante un problema de clasificación la salida será un vector de n resultados que indique la probabilidad que tiene la entrada de pertenecer a cada una de las clases, siendo n el número de clases. Las funciones de salida que se usan en estos problemas son la función sigmoid y softmax. La función sigmoid hace que cada uno de los elementos del vector de salida tenga un valor entre 0 y 1, el cual indicará el grado de pertenencia a cada una de las clases. Para clasificar una entrada se le asignará la etiqueta de la clase que sobrepase un umbral, como puede ser 0.75. Esto provoca que para una entrada pueda haber varias etiquetas, por lo tanto esta función no es útil para nuestro problema. Por su parte, la función softmax hace que los valores del vector de salida estén entre 0 y 1 y la suma total sea 1. De esta forma por cada entrada solo se asignará una clase, la que mayor probabilidad tenga. Los problemas seleccionado tienen clases disjuntas por lo que la función recomendada es la función softmax.

Partiendo de nuestra función de salida tendremos que elegir como calcular el error de nuestra red. Sabemos que estamos en un problema de clasificación por lo que la forma en la que tenemos que ajustar nuestro modelo es mediante la máxima verosimilitud. La máxima verosimilitud es un método para ajustar los parámetros de un modelo para que pueda hacer predicciones cuya distribución de probabilidad coincida con la distribución de probabilidad de las etiquetas. Consiste en un problema de optimización en el que se intenta hacer mínima la diferencia entre ambas distribuciones de probabilidad. Para calcular esta diferencia se usa la función de error de la entropía cruzada. Por lo tanto, en el framework que vamos a usar, que es PyTorch (en el siguiente apartado se aclara su elección) tendremos que encontrar la función que calcule la entropía cruzada. Existen dos alternativas: corss entropy loss y negative log likelihood. Ambas calculan la entropía cruzada pero se diferencian en el tipo de entrada que aceptan. La cross entropy loss espera que nuestra red ofrezca los resultados sin transformación, es decir, que la salida no sean probabilidades. Por su parte la negative log likelihood si que espera que la salida de nuestra red sean probabilidades. Como nuestra red usa la función softmax, las salidas están codificadas como probabilidades, la función negative log likelihood será la función de error que tendremos que usar.

Métricas

Una vez entrenado el modelo se deberá de evaluar que tan bueno es. Para ello se hará uso del conjunto de test, el cual no se ha usado durante ningún momento del entrenamiento.

Un clasificador es más bueno que otro si es capaz de ajustar mejor la distribución de probabilidad de la muestra. Pero, ¿cómo podemos medir este ajuste?. Una aproximación es: dado un conjunto de test, medir que porcentaje del total ha conseguido clasificar correctamente, es decir, la precisión de nuestro modelo. Cuanto más muestras clasifique correctamente, mejor será nuestro modelo. Pero esto no siempre es así. En el caso de problemas cuyas clases no estén equilibradas, es decir, que no haya el mismo número de muestras de cada clase, podemos equivocarnos al tomar esta medida, pues se tendrá que tener en cuenta esta descompensación. Ejemplo de ello son los problemas de predicción de cáncer de mama. La mayoría de las muestras son muestras negativas, y la minoría positivas, por lo que si un modelo solo predijese valores negativos tendría una precisión muy alta si lo medimos erróneamente. En los dos problemas que hemos seleccionado todas las clases están equilibradas por lo que no tenemos que preocuparnos por compensar el desequilibrio en el cálculo de la métrica. Por lo tanto, la métrica que usaremos será el accuracy (precisión).

$$precision = \frac{Muestras\ clasificadas\ correctamente}{Total\ de\ muestras} x100 \tag{4.1}$$

4.2. Implementación

A continuación, voy especificar cuales son las herramientas que voy a emplear en la implementación.

El lenguaje que he seleccionado para la implementación de este estudio es python. Este lenguaje es uno de los más usados a nivel global, y el más usado en el ámbito del machine learning [1]. El motivo de su amplia utilización en machine learning es debido a su fácil depuración, vital en este campo de desarrollo, y a la gran cantidad de frameworks que facilitan las tareas de desarrollo, implementación y despliegue. Entre los frameworks más famosos y que se centran en el desarrollo de redes neuronales nos encontramos TensorFlow [2] y PyTorch [3]. Tensorflow fue desarrollado desde un inicio para tareas de visión por computador, permitiendo programar modelos de redes neuronales con pocas lineas de código. Tiene un gran rendimiento y un gran conjunto de herramientas de desarrollo, sin embargo, tiene el inconveniente de que para tareas tan importantes como el debugging es necesario el uso de herramientas específicas, tfdbg. Por su parte, PyTorch también es un framework muy eficiente, pero es más fácil de debuggear (no se necesita aprender ninguna herramienta específica) debido a que hace uso de lo que se

[1] `https://www.kaggle.com/kaggle-survey-2021`
[2] `https://www.tensorflow.org/`
[3] `https://pytorch.org/`

conoce como grafo computacional dinámico, mientras que TensorFlow usa una aproximación estática [38]. Además, PyTorch es el framework que más se suele utilizar en investigación, ejemplo de ello es que todos los artículos científicos que he seleccionado que estudian alternativas del backpropagation realizan su implementación en PyTorch. En conclusión, por su accesibilidad y por la disponibilidad de los algoritmos que voy a estudiar, el framework que he seleccionado para realizar mi proyecto es PyTorch.

Además de un framework de desarrollo de redes neuronales tendré que seleccionar las bibliotecas necesarias para obtener gráficos, realizar cálculos matriciales e implementar los algoritmos que he seleccionado. Con respecto a las gráficas, la biblioteca más utilizada es matplotlib [4]. Tiene una gran cantidad de herramientas para crear gráficos entendibles y de manera sencilla, lo que facilitará la presentación de los resultados. Para el manejo de vectores y matrices usaré la famosa biblioteca numpy [5], debido a su gran eficiencia y facilidad de uso. Finalmente, para la implementación de los algoritmos que han sido seleccionados para el estudio se usará los repositorios de github que se especifican en los respectivos artículos científicos. En el caso del FeedBack Alignment he usado el sitio web PapersWithCode [6] para encontrar un repositorio donde se implementara dicho algoritmo. Como he mencionado en el párrafo anterior, todas las implementaciones han sido realizadas en PyTorch por lo que no habrá problemas de incompatibilidad. Los repositorios de los algoritmos son los siguientes: HSIC [7], Synthetic Gradients [8] y FeedBack Alignment [9].

Todas las experimentaciones mencionadas se han realizado haciendo uso de un conjunto de scripts en python y bash. A continuación se explicará la estructura de directorios empleadas y la función de cada directorio.

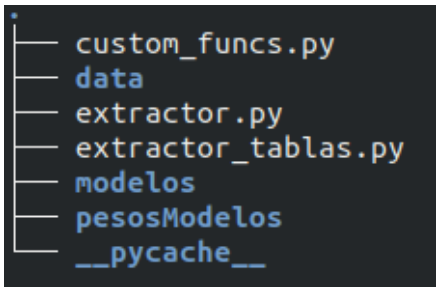


Figura 4.3: Directorio raíz

[4] https://matplotlib.org/
[5] https://numpy.org/
[6] https://paperswithcode.com/
[7] https://github.com/choasma/HSIC-Bottleneck
[8] https://github.com/koz4k/dni-pytorch
[9] https://github.com/jsalbert/biotorch

El archivo custom_funcs es un script de python en el que se encuentran todas las funciones comunes a los algoritmos, como son las funciones de training, test, carga de datos, etc. Los scripts extractor y extractor tablas son scripts para graficar los resultados obtenidos con la experimentación. Pasemos a analizar los directorios. El directorio data contiene los conjuntos de datos a usar en este estudio: MNIST y FMNIST. El directorio pesosModelos contiene los pesos de las redes neuronales que nos ha interesado almacenar (redes neuronales sin cuantificar). Y por último, el directorio modelos contiene todos los scripts en los que se han programado los algoritmos y los scripts de bash para la automatización de las pruebas. Veamos que contiene este directorio.

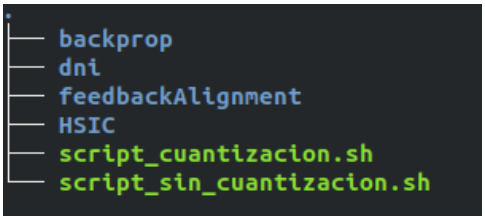


Figura 4.4: Directorio modelo

El script sin cuantización es un script de bash que se usa para ejecutar los modelos sin ningún tipo de cuantificación, mientras que el script con cuantificación ejecuta los algoritmos aplicando la cuantificación. Cada uno de estos scripts son totalmente modificables pues se han parametrizado en su totalidad para que su alteración sea rápida, sencilla y libre de errores. Con respecto a los directorios, cada uno contiene los scripts con su algoritmo correspondiente y los datos obtenidos que se obtienen con la experimentación. A continuación, voy a mostrar como es la estructura de estos directorios. Para ello solo analizaré uno de ellos ya que el resto son iguales, a excepción de HSIC que contiene carpetas extras debido a la librería usada para su elaboración.

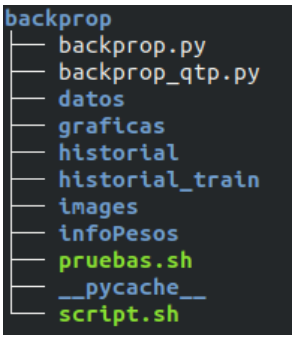


Figura 4.5: Directorio Backpropagation

En este directorio encontramos un scipt para cada versión del algoritmo, la cuantificada y la que no tiene cuantificación. Con cada uno de estos script se ejecuta el algorimto y los resultados se almacenan en los directorios que se explicarán ac continuación. Los scripts de bash son usados para automatizar la experimentación, el script script.sh se especifica la arquitectura de la red neuronal a usar en la experimentación. Esta estructura se especifica como parámetros del script pruebas.sh. Este script se emplea para especificar los parámetros con los que realizar la experimentación: número de bits, número de épocas, conjunto de datos, nivel cuantificación, función de cuantificación, etc. Por su parte los directorios se usan para guardar la información extraída de la experimentación.

- **datos**: se almacena la precisión alcanzada en cada experimentación junto a los parámetros establecidos para la misma.
- **graficas**: gráficas extraídas con los scripts extractor y extractor tablas.
- **historial**: archivos con la evolución de la precisión sobre el conjunto test, durante el entrenamiento de cada una de los modelos.
- **historial_train**: archivos con la evolución de la precisión sobre el conjunto de entrenamiento, durante el entrenamiento de cada una de los modelos.
- **images**: graficas con la evolución de la precisión sobre el conjunto test, durante el entrenamiento de cada una de los modelos.
- **infoPesos**: archivos con la evolución de los pesos de los modelos durante el entrenamiento, se almacenan los valores máximos y mínimos de cada capa.

Todos estos archivos se pueden consultar en mi GitHub [10]

4.3. Hipótesis iniciales

En este apartado voy a establecer unas hipótesis iniciales acerca de qué espero de la experimentación. Para ello partiré de los conocimientos adquiridos durante el estudio de los algoritmos seleccionados y de los métodos de cuantificación que voy a usar.

- **H1: Los algoritmos sin cuantificación alcanzan las mismas precisiones**: Lo primero que se hará es ver los resultados que tiene cada uno de los modelos entrenados con los distintos algoritmos

[10] `https://github.com/pacoapm/TFG-Francisco-Jose-Aparicio-Martos`

de entrenamiento sin ningún tipo de cuantificación. Según los resultados que ofrecen los artículos científicos todos deberían de llegar a resultados parecidos a los del backpropagation. Sin embargo, en los estudios se presentan arquitecturas grandes, en especial en el synthetic gradient que se usan para modelos muy profundos, por lo que en las arquitectura más pequeñas su comportamiento será más impredecible.

- **H2: Límite de cuantificación en torno a 6 o 7 bit**: Tras obtener los modelos sin cuantificar, se aplicaran las cuantificaciones y se observará como afecta la misma al entrenamiento. Según explican en el estudio de donde se han sacado las funciones de cuantificación, los modelos se pueden comprimir satisfactoriamente hasta los 6 bits, pasada esta barrera (2-3-4-5) los modelos pierden rendimiento de manera considerable. Teniendo en cuenta que estas cuantificaciones se realizaron sobre modelos ya entrenados, aplicar la cuantificación durante el entrenamiento puede llegar a producir peores resultados ya que durante el proceso de entrenamiento se pierde información por la cuantificación. Por lo tanto la barrera a partir de la cual las cuantificaciones no sean rentables aplicarlas tendrá que estar en torno a los 6 o 7 bits.

- **H3: ASYMM y SYMM ofrecen mismo rendimiento**: Los modelos cuantificados por una función u otra tendrán que ofrecer resultados parecidos.

- **H4: Mejor algoritmo: HSIC**: De los algoritmos seleccionados el que mejor puede resistir la cuantificación es el HSIC, pues este algoritmo va calculando el gradiente capa por capa, sin propagación del error, por lo tanto, a menor cantidad de operaciones encadenadas, menor será la imprecisión numérica que introduce la cuantificación.

- **H5: Parámetro diferencial arquitecturas: Anchura**: Con respecto a las arquitecturas, pienso que estructuras más grandes pueden compensar la pérdida de precisión de las sinapsis, siendo la anchura de las capas el factor diferencial. En algoritmos como el backprop no interesa tener redes muy profundas debido al arrastre del error, por lo que será preferible tener arquitecturas menos profundas pero más anchas.

Capítulo 5

Valores experimentales

En este capítulo se expondrán los resultados de los experimentos realizados, creando una apartado por algoritmo. Se estudiará: como afectan la cantidad de bits al proceso de cuantificación, si hay diferencia de rendimiento final al aplicar distintas funciones de cuantificación (ASYMM 2.1 y SYMM2.2), si aplicar la cuantificación a nivel local mejora el nivel global, y como afecta la anchura y la profundidad al rendimiento de las redes cuantificadas. Cada uno de los experimentos realizados se han ejecutado una única vez debido a los límites de tiempo. En total se han realizado 4 algoritmos x 2 niveles cuantificacion x 7 cantidades bits x 2 funciones cuantificacion x 6 arquitecturas = 672 experimentos. Cada uno de estos experimentos tarda de media unos 7 minutos en ejecutarse, por lo que en el tiempo total que se ha empleado en las ejecuciones es de: 672 experimentos x 5.5 minutos = 4704 minutos = 78.4 horas = 3.26 días. Además de estos experimentos, se han realizado una pruebas extras para ver que los resultados reflejados en la experimentación son representativos y no tienen mucha varianza dependiendo de la semilla inicial elegida. Tanto para las experimentaciones como para mostrar los resultados, se han creado scripts en bash y python para automatizar todos los procesos.

5.1. Backpropagation

El Backpropagation es nuestro algoritmo base de comparación. En este apartado veremos como afecta la cuantificación a este algoritmo.

5.1.1. Arquitectura Base

La comparación entre las funciones de cuantificación se ha realizado sobre el modelo base de 1 capa y 4 neuronas por capa.

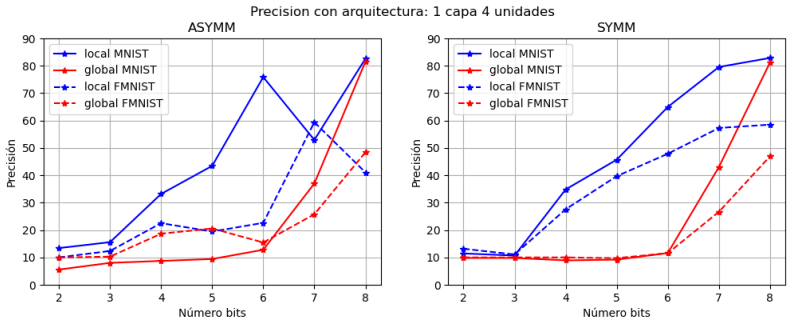


Figura 5.1: ASYMM vs SYMM

En esta dos gráficas se representan el rendimiento del modelo base usando la función de cuantificación ASYMM (gráfica de la izquierda) y SYMM (gráfica de la derecha). Cada una de las funciones se ha probado variando la cantidad de bits a usar y el nivel al que se aplica la cuantificación.

ASYMM

Observando los resultados de los modelos entrenados con ASYMM, podemos apreciar como hay una tendencia a mejorar la precisión conforme aumenta el número de bits usados. Si nos fijamos en las precisiones que alcanzan los modelos sobre los conjuntos de datos, se consiguen mejores resultados sobre MNIST que sobre FMNIST. Esto se debe a que (como habíamos definido en 4.1.2) el problema FMNIST es más complejo y por lo tanto, más difícil de aprender. Centrándonos en el problema MNIST, vemos que la cuantificación local es superior a la global, por lo tanto, la restricción de valores afecta negativamente al rendimiento de los modelos cuantificados. A partir de los 6 bits, los modelos locales empiezan a conseguir precisiones razonables, sin embargo, el modelo de 7 bits tiene peor desempeño que el modelo de 6 bits. ¿Qué ha podido ocurrir? Fijémonos en los entrenamientos de los modelos de 6, 7 y 8 bits.

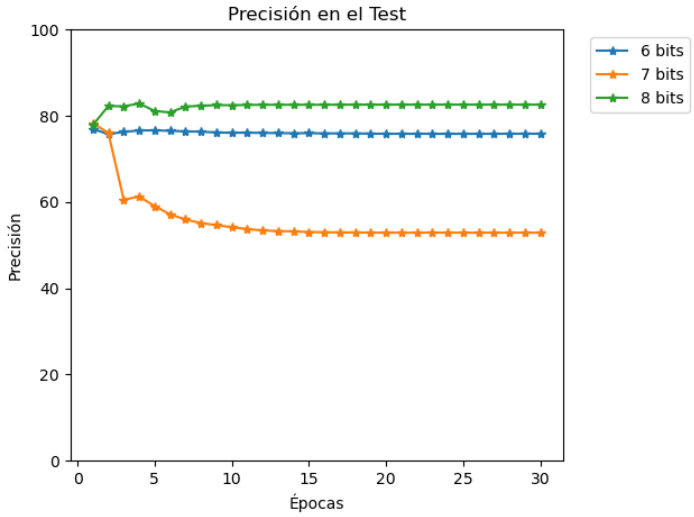


Figura 5.2: Entrenamiento de los modelos de 6, 7 y 8 bits

Todos los modelos en su primera época consiguen alcanzar un precisión ligeramente alta, en torno al 80 %. Conforme pasan las épocas, los modelos de 6 y 8 bits se quedan en este óptimo, pero el modelo de 7 bits diverge y se estanca en un óptimo peor del que no puede salir. Este fenómeno se puede dar debido al ruido que introduce el proceso de cuantificación. Cuanto más se reduce la precisión numérica, mayor es este ruido. Sin embargo, el modelo de 6 bits no llega a divergir. Este fenómeno se puede dar debido a que al tener menor precisión, el ruido introducido es de menor tamaño que la resolución mínima numérica. Por lo tanto, no perturba el resultado final, actuando como un regulador.

Volviendo a la gráfica 5.1 vemos que sobre el conjunto FMNIST se producen las mismas tendencias que en MNIST. Los modelos con cuantificación local consiguen mejores resultados que los que usan la cuantificación global. Sin embargo, con esta arquitectura, los modelos cuantificados no son capaces de conseguir resultados razonables, necesitando la máxima precisión numérica (8 bits) para llegar a rozar el 50 % de precisión. Por lo tanto, se necesita una arquitectura de mayor tamaño que ofrezca una solución mas compleja.

SYMM

Al igual que ocurre en los modelos entrenados con la función ASYMM, conforme se aumenta número de bits también aumenta la precisión de los modelos. Con respecto a los conjuntos de datos y niveles de cuantificación, también sucede lo mismo. Se consiguen mejores resultados en MNIST de-

bido a su mayor sencillez. La cuantificación a nivel local es superior a la global debido a que no se reduce el espacio de búsqueda de soluciones. Sin embargo, existe una ligera diferencia entre ambas funciones. Y es que la mejora conforme se aumenta la precisión numérica es más estable que en los modelos que emplean ASYMM. De aquí podemos intuir que los procesos de entrenamiento con cuantificación simétrica tienden a ser menos ruidosos.

5.1.2. Arquitecturas grandes

A continuación, se van a mostrar las precisiones alcanzadas por los modelos con mayor profundidad y anchura.

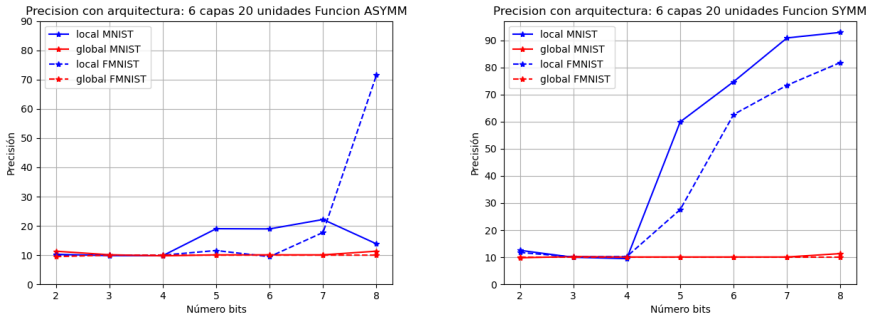


Figura 5.3: Arquitectura 6 capas 20 unidades

En esta arquitectura se han aumentado profundidad para ver como

Con esta arquitectura podemos ver como afecta un aumento de la profundidad a las redes cuantificadas. Para ambas funciones, la cuantificación global es inviable. Sin embargo, la cuantificación local si que difiere de una una función a otra. Para la función ASYMM podemos ver que en ambas funciones se consiguen resultados pésimos. A excepción de cuando se usan 8 bits en la función FMNIST, que se alcanza un óptimo del 70 %. Este valor se ha podido dar simplemente porque en la búsqueda a dado con un buen óptimo y se ha quedado ahí estancado.

Por su parte, los modelos cuantificados con SYMM a nivel local, consiguen resultados decentes en ambos problemas a partir de los 6 bits.

Visto como afecta un aumento significativo de la profundidad a los modelos cuantificados, vamos a explorar como afecta el aumento de la anchura. Para ello haremos uso de modelos con 2 y 3 capas de profundidad a las que aumentaremos en anchura para ver si se mejoran o no los resultados.

Empezaremos por los modelos con 2 capas ocultas.

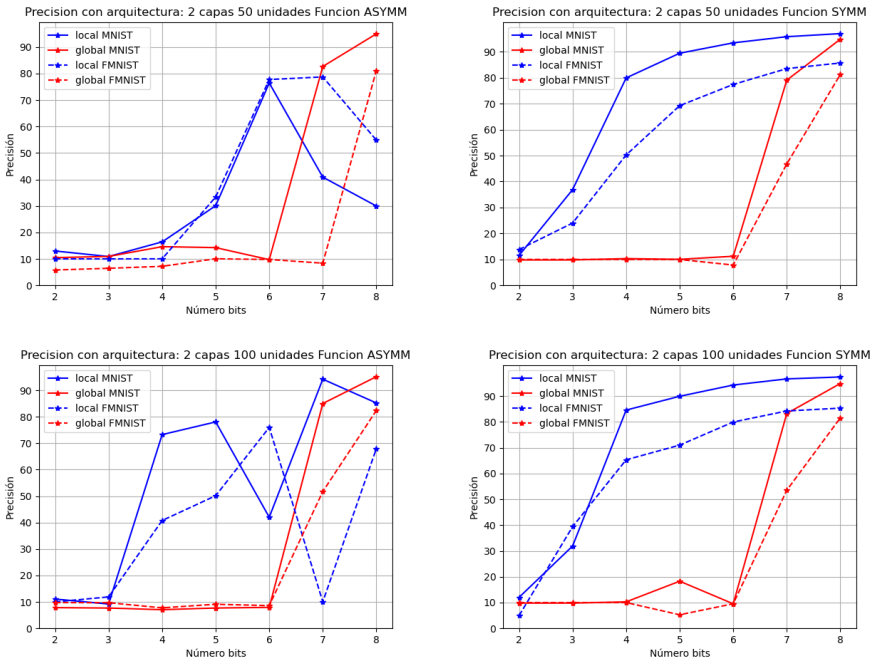


Figura 5.4: Arquitectura: 2 capas ocultas

Con este aumento de anchura se pueden ver claras diferencias entre ambas funciones de cuantificación. Los modelos que usan la función ASYMM tienen una clara inestabilidad en la búsqueda de óptimos. Aumentar el número de bits aumenta la capacidad de los modelos para encontrar mejores óptimos, pero no asegura encontrarlos. Esto se puede apreciar en los picos de bajada de las gráficas de la izquierda. Por su parte los modelos que usan las funciones de cuantificación SYMM, son mucho más estables y a mayor número de bits, mayor precisión tienen los modelos. Se sigue cumpliendo que la cuantificación local es superior a la global. Con cuantificación local a partir de los 4 bits se consiguen modelos decentes, mientras que para la cuantificación global se necesitan 8 bits para tener resultados decentes. Con respecto a la diferencia de profundidades vemos que hay una ligera superioridad a los modelos con anchura de 100 unidades. Siendo los modelos con 4 bits donde más se nota esta diferencia.

Ahora vamos a ver lo que ocurre si añadimos una capa oculta más a los modelos.

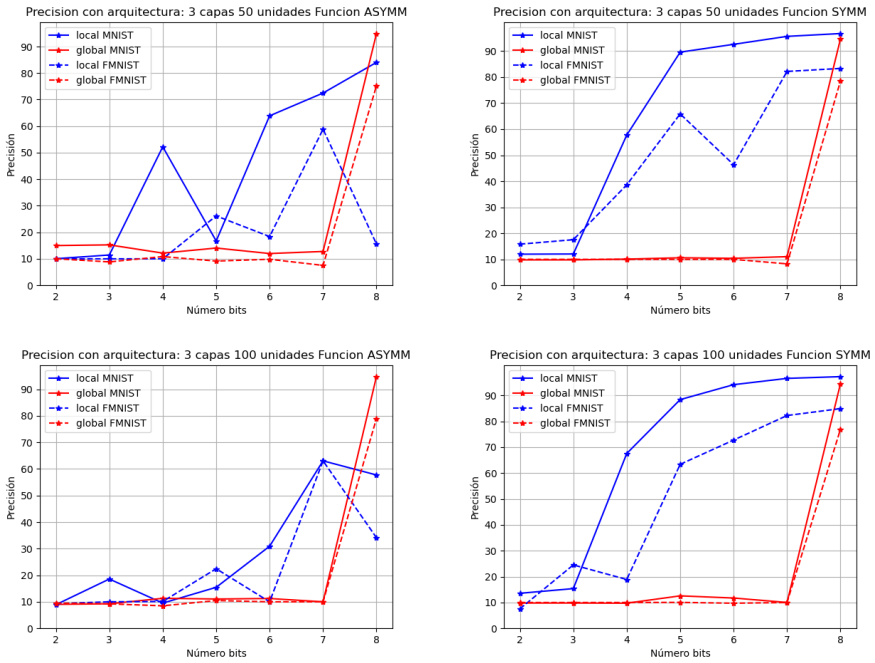


Figura 5.5: Arquitectura: 3 capas ocultas

En los modelos cuantificados con la función ASYMM, sigue ocurriendo lo mismo que en los modelos con dos capas. Se mejora la precisión al aumentar el número de bits, pero no siempre ocurre. Debido al ruido, los modelos divergen y no son capaces de encontrar óptimos. Por su parte, los modelos cuantificados mediante SYMM localmente siguen teniendo mejores resultados. A mayor número de bits, mayor precisión. El modelo con anchura de 100 unidades sigue siendo superior al de 50 unidades, obteniendo precisiones más elevadas cuando se disminuye el número de bits. Por su parte, los modelos con cuantificación global necesitan 8 bits para poder tener resultados decentes.

5.2. HSIC

En este apartado vamos a analizar el comportamiento del algoritmo HSIC.

5.2.1. Arquitectura base

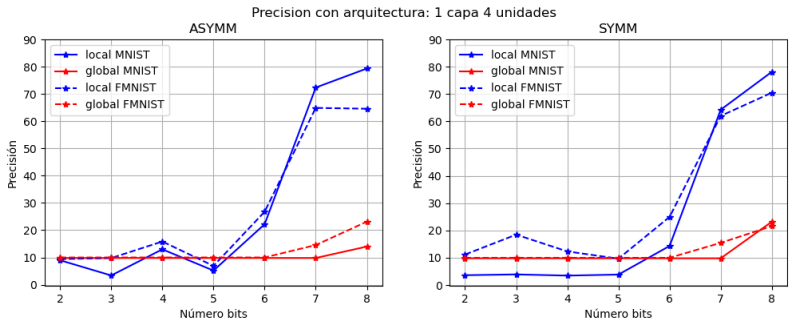


Figura 5.6: ASYMM vs SYMM

A diferencia de lo que ocurría con el algoritmo de Backpropagation, las gráficas de ambas funciones son prácticamente idénticas. En ambas se ve que con el aumento de la precisión numérica consigue mejorar la precisión de los modelos. Debido a la poca diferencia que hay entre las funciones, voy a realizar el análisis del comportamiento del modelo base sin diferenciar la función de cuantificación a usar.

Con respecto a los problemas tratados, se vuelve a cumplir que los mejores resultados se obtienen en MNIST, debido a su sencillez. Donde si que hay una mayor diferencia es en el nivel al que se aplica la cuantificación. En este algoritmo la cuantificación global hace imposible el aprendizaje. Además, la cuantificación local necesita de una precisión de 7 bits (mayor a la necesitada por el Backpropagation) para tener resultados un poco más decentes. Para intentar comprender por qué este algoritmo se ve tan afectado por la cuantificación global vamos a consultar los máximos y mínimos alcanzados por los pesos de los modelos.

Número bits	**Varianza mínimo**	**Varianza máximo**	**Peso mínimo**	**Peso máximo**
6	0.1061	0.7506	-1.0	1.0
7	0.0716	0.8308	-1.0	1.0
8	0.0808	0.8026	-1.0	1.0

Cuadro 5.1: Información de los pesos cuantificados

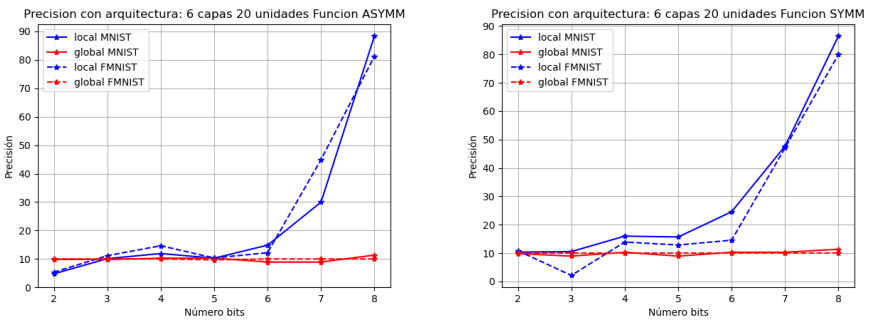
Número bits	Varianza mínimo	Varianza máximo	Peso mínimo	Peso máximo
6	7786.2084	11 660.1328	−1250.5876	1340.1875
7	4331.5791	9098.9863	−1002.5652	1239.8550
8	5846.4846	8003.4925	−984.1152	1209.6030

Cuadro 5.2: Información de los pesos sin cuantificar

Como podemos apreciar, en este algoritmo, los pesos tienden a tener valores bastantes grandes, por lo tanto, si fijamos el rango de valores entre -1 y 1, el algoritmo no será capaz de alcanzar óptimos.

Arquitecturas grandes

Al igual que en el Backpropagation, primero vamos a analizar como afecta el aumento de la profundidad al modelo.



Al igual que en el modelo base, no hay mucha diferencia entre ambas funciones de cuantificación. Y como ocurría con el Backpropagation, profundidades más elevadas hacen imposible la aplicación de la cuantificación a nivel global. Por su parte, los modelos con cuantificación local, haciendo uso de 8 bits, consiguen mejorar los resultados del modelo base. Por lo tanto, ese aumento de profundidad permite mejorar los resultados de la red a nivel local, mientras que el nivel global es inviable.

Ahora vamos a analizar como afecta al algoritmo un aumento considerable de la anchura del modelo, empezando por los modelos con 2 capas ocultas y 50 y 100 neuronas.

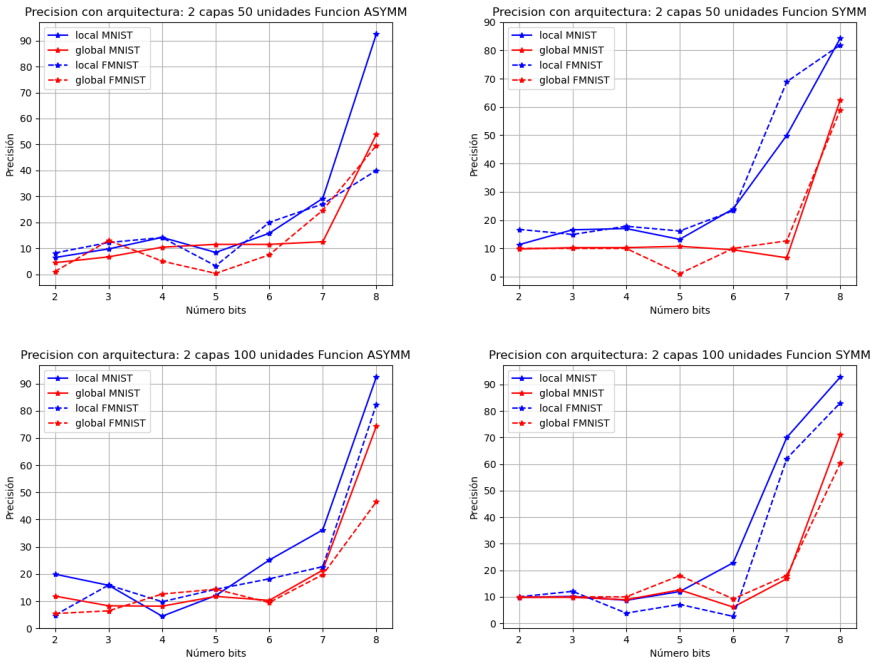


Figura 5.8: Arquitectura: 2 capas ocultas

Con arquitectura más grandes, podemos observar que los modelos que usan la función de cuantificación SYMM consiguen mejores resultados. Esta diferencia se nota cuando se llega a los 8 bits, pues por debajo de este umbral, el algoritmo no es capaz de alcanzar buenos resultados. Además, con esta función, se consiguen prácticamente las mismas precisiones en ambos problemas. Con respecto al aumento de la anchura, vemos que la diferencia se nota en los 8 bits. En este caso, los resultados alcanzados por la cuantificación local superan en un 20 % a los alcanzados por la cuantificación global. Y por su parte, la cuantificación global, consigue mejores resultados con el aumento de la anchura.

Ahora vamos a ver lo que ocurre cuando añadimos una capa extra.

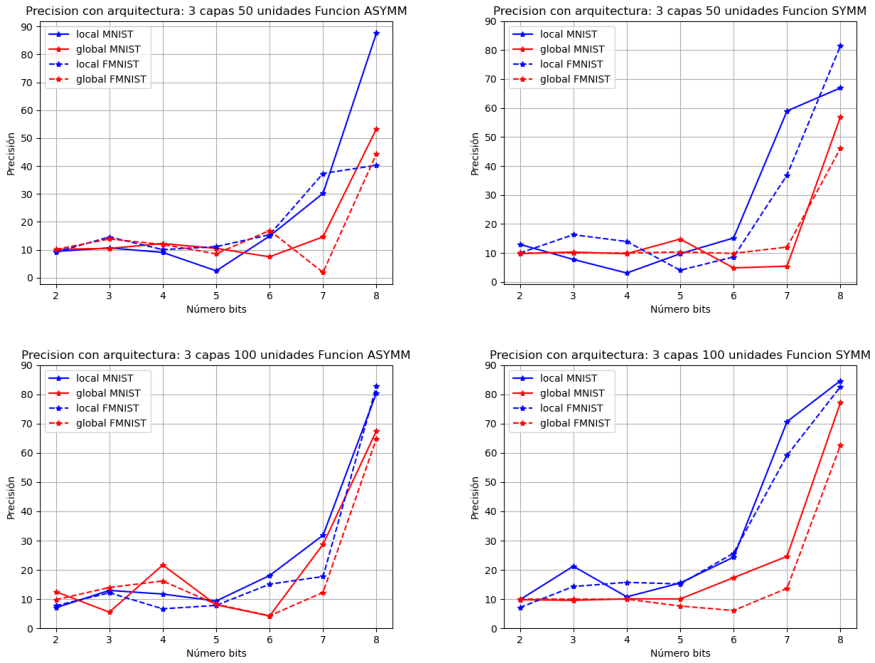


Figura 5.9: Arquitectura: 2 capas ocultas

Estos modelos consiguen resultados muy parecidos a los alcanzados por los modelos de 2 capas. Para los modelos que usan la función ASYMM, los resultados son prácticamente idénticos, siendo la mayor diferencia: la mejora de precisión del modelo de 8 bits con cuantificación global sobre FMNIST, consiguiendo mejorar un 20 %. Es en los modelos que se han cuantificado con SYMM donde podemos encontrar más diferencias. En los modelos con anchura de 50 neuronas, los modelos con cuantificación global pierden precisión en comparación con los modelos de una capa oculta menos. Además, con cuantificación local, se ha conseguido mejores resultado sobre FMNIST que con MNIST, esto se ha podido dar por el sobreentrenamiento. Aumentar la complejidad de la red ha permitido mantener la precisión en el problema FMNIST, pero provoca un sobreajuste en el problema MNIST. Cuando se aumenta la anchura a 100 unidades por capa, se consigue mejorar la precisión de todos los modelos. En los modelos cuantificados con ASYMM: se consigue mejorar los resultados de los modelos cuantificados a nivel local, se aumenta la precisión en FMNIST y en MNIST decae ligeramente la precisión. Por su parte, los modelos cuantificados con SYMM consiguen mejores resultados cuando usan 7 bits. Con 8 bits los resultados son casi idénticos, consiguiendo una ligera mejora en los modelos cuantificados globalmente.

5.3. FeedbackAlignment

A continuación, analizaremos el comportamiento del algoritmo de FeedbackAlignment.

5.3.1. Arquitectura base

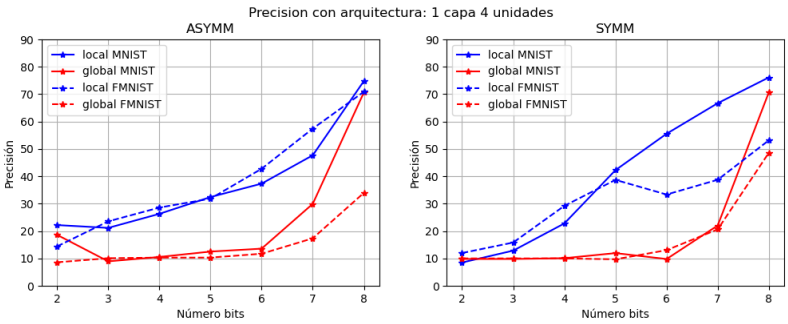


Figura 5.10: ASYMM vs SYMM

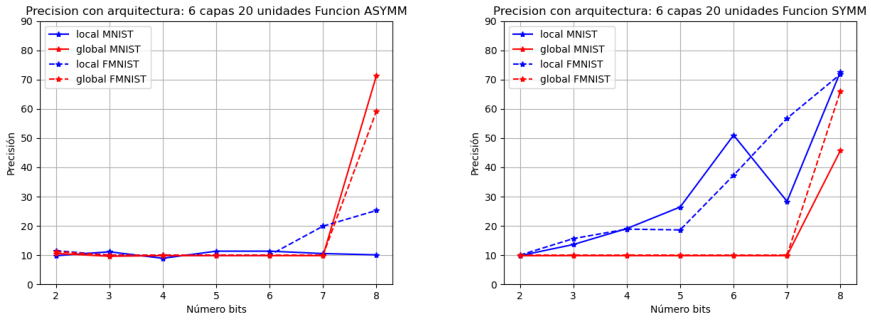
Empecemos por analizar el comportamiento de los modelos cuantificados con ASYMM. Para los modelos locales, vemos que este algoritmo ofrece precisiones muy parecidas en ambos problemas. Conforme aumenta el número de bits aumenta la precisión obtenida, llegando al 70 % en ambos problemas con los 8 bits de precisión. Por su parte, en los modelos con cuantificación global, la precisión obtenida con 8 bits en MNIST iguala prácticamente a la alcanzada por los modelos locales, mientras que en FMNIST, no se consigue superar el 40 % de precisión.

Ahora pasamos a lo modelos cuantificados con la función SYMM. Los modelos cuantificados con esta función mejora los resultados obtenidos en MNIST, consiguiendo mejores resultados con menor cantidad de bits. Sin embargo, en FMNIST, los resultados de los modelos cuantificados localmente disminuyen, cayendo casi al 50 % con 8 bits, mientras que los cuantificados globalmente aumentan llegando prácticamente a esta precisión. Por lo tanto, con el modelo base, ASYMM funciona mejor para FMNIST, mientras que para MNIST es mejor usar SYMM.

5.3.2. Arquitecturas grandes

Analizado el comportamiento del modelo base, toca analizar como afecta el aumento de la profundidad y anchura de las redes.

Como en los anteriores algoritmos, lo primero que vamos a analizar es como afecta aumentar la profundidad de la red.



Viendo las gráficas podemos apreciar que los resultados no son muy positivos. Empecemos por los modelos cuantificados por ASYMM. La cuantificación local es inviable, el ruido introducido por el algoritmo hace imposible superar el 30 % de precisión, lo que supone un resultado de un clasificador prácticamente aleatorio. La cuantificación local, en cambio, consigue con 8 bits precisiones del 70 % para MNIST y 60 % para FMNIST. No son resultados muy buenos pero superan considerablemente a los locales. Una posible razón de esta mejora, es que la restricción sobre los pesos haya actuado como regulador, permitiendo alcanzar mejores resultados sin divergir. Fijémonos ahora en los modelos cuantificados con la función SYMM. Ahora los modelos locales consiguen los mejores resultados. Llegado a los 8 bits se consigue un 70 % de precisión en ambos problemas. Los modelos globales por su parte, no consiguen precisiones muy elevadas. Con 8 bits se consigue casi un 70 % de precisión en FMNIST. Este valor no hay que tenerlo muy en cuenta ya que con 1 bit menos no se ha conseguido pasar del 10 % de precisión. Por lo que posiblemente ese casi 70 % se haya podido alcanzar debido a factores aleatorios, como el ruido introducido por la cuantificación.

A continuación, vamos a ver como afecta el aumento de la profundidad a este algoritmo. Comenzaremos con las arquitecturas de 2 capas ocultas.

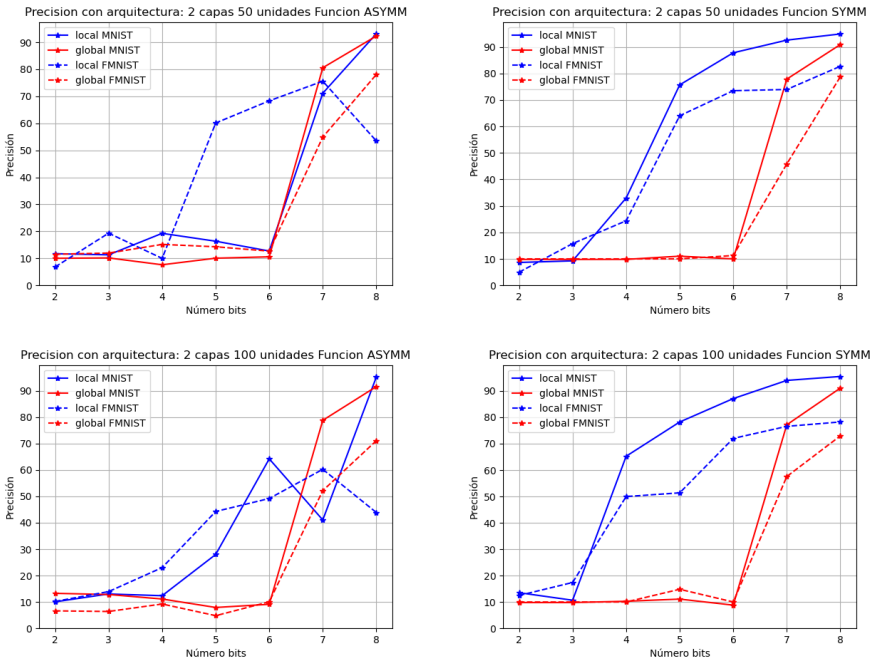


Figura 5.12: Arquitectura: 2 capas ocultas

A primera vista, se puede apreciar que este algoritmo es mejor en redes más anchas que profundas. Vamos primero a analizar los modelos con anchura de 50 y después las de 100.

Viendo ambas gráficas, podemos ver que el aumento del número de bits supone una mejora en la precisión. A excepción del modelo local con 8 bits en el problema FMNIST. En este caso lo más seguro es que debido al aumento de precisión, el ruido introducido sea susceptible de alterar el comportamiento de la red, y por lo tanto, provocar que diverja y no alcance óptimos. El modelo global de 8 bits consigue mejores resultados en FMNIST, posiblemente debido a la restricción de pesos, que funciona como regulador. En MNIST, sin embargo, ambos niveles de cuantificación consiguen resultados en torno al 90 % con 8 bits de cuantificación. Si reducimos la precisión a 7 bits se consiguen resultados más o menos elevados, siendo el modelo cuantificado globalmente (80 % precisión) ligeramente superior al cuantificado localmente (70 % precisión).

Centrémonos ahora en los modelos cuantificados con SYMM. Todos los modelos mejoran al aumentar el número de bits, llegando a alcanzar el 90 % de precisión en MNIST y el 75-80 % en FMMIST. Con 8 bits se consiguen los mismos resultados que en los modelos cuantificados con ASYMM, con la diferencia de que: el modelo cuantificado localmente consigue mejorar hasta alcanzar el 75 % de precisión. Donde hay una clara mejora es en los modelos

cuantificados localmente con 8 bits, superando el 70 % en ambos problemas a partir de los 6 bits. Por su parte, los modelos cuantificados localmente siguen necesitando mayor número de bits (7 bits en MNIST y 8 en FMNIST) para alcanzar valores buenos.

Aumentando la anchura a 100 bits podemos ver que las precisiones mejoran. Para los modelos cuantificados con ASYMM vemos como por debajo de los 8 bits los resultados mejoran, pero siguen siendo resultados bastante pésimos. Con 8 bits de precisión, los resultados se mantienen. Si pasamos a los modelos cuantificados con SYMM podemos apreciar una mejoría. Los modelos entre 5-8 bits mantienen las mismas precisiones, mejorando ligeramente. Es con 4 bits donde se nota una mayor mejoría, aunque siguen sin llegar a precisiones muy altas.

Ahora veamos que es lo que ocurre cuando aumentamos la profundidad de la red en 1 capa.

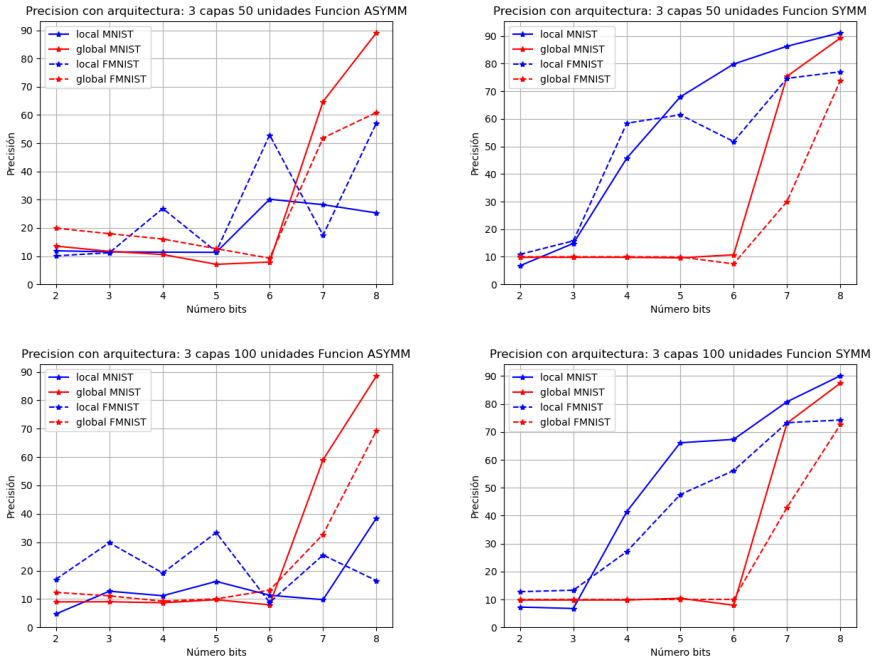


Figura 5.13: Arquitectura: 3 capas ocultas

En los modelos que usan 50 unidades, más en específico los cuantificadas con ASYMM, podemos ver un empeoramiento de los resultados. Con los 8 bits se consiguen los mejores resultados, siendo los cuantificados globalmente los que mayor precisión alcanzan. En MNIST, se mantiene la precisión con el modelo cuantificado globalmente, pero el cuantificado localmente decae drásticamente a un 25 %. En FMNIST bajan los modelos cuantificados global

y localmente. Pasando a los modelos cuantificados con la función SYMM, vemos que el comportamiento no varía mucho con respecto a los modelos con una capa menos. Cuando aumentamos la anchura a 100 unidades, podemos ver que los modelos cuantificados con ASYMM localmente empeoran su rendimiento, mientras que globalmente con 8 bits, mejoran ligeramente y con 7 bits empeoran. Por su parte, los cuantificados con SYMM no sufren gran cambio, consiguiendo precisiones muy parecidas.

5.4. Synthetic gradients

Finalmente toca analizar el algoritmo de synthetic gradients.

5.4.1. Modelo base

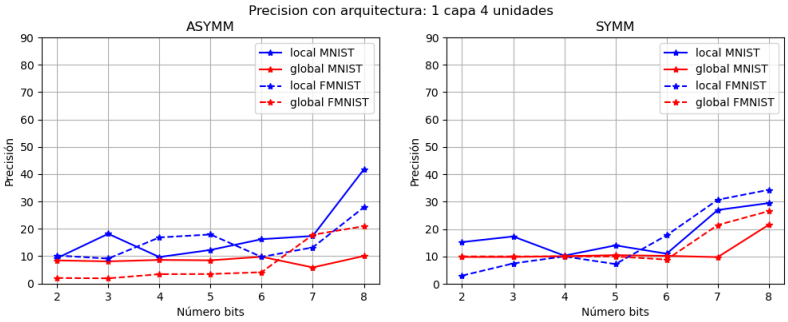
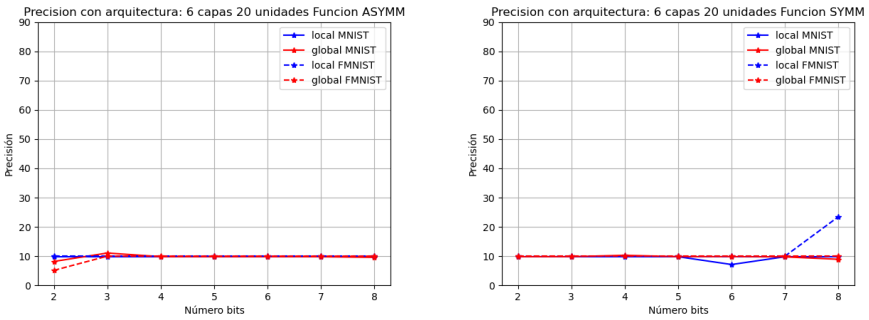


Figura 5.14: Arquitectura base

Viendo ambas gráficas, podemos apreciar que este algoritmo es incapaz de resolver los problemas. Aún usando 8 bits, solo es capaz de llegar al 40 % de precisión en MNIST. Por lo tanto, con un modelo tan pequeño, aplicar synthetic gradient es inviable.

5.4.2. Arquitecturas grandes

Veamos ahora si aumentando el tamaño de la red se consiguen mejores resultados. Empecemos por aumentar la profundidad.



Con este aumento de profundidad vemos que empeoran aun más los resultados. Todos los modelos obtienen un 10 % de precisión, lo que es equivalente a una clasificación aleatoria. El 20 % alcanzado en FMNIST no tiene mucho que destacar, ya que esta precisión también es propia de un clasificador aleatorio. Por lo tanto, con esta arquitectura también es imposible aplicar este algoritmo de entrenamiento. Veamos si con los modelos con más anchuras se consigue alguna mejora.

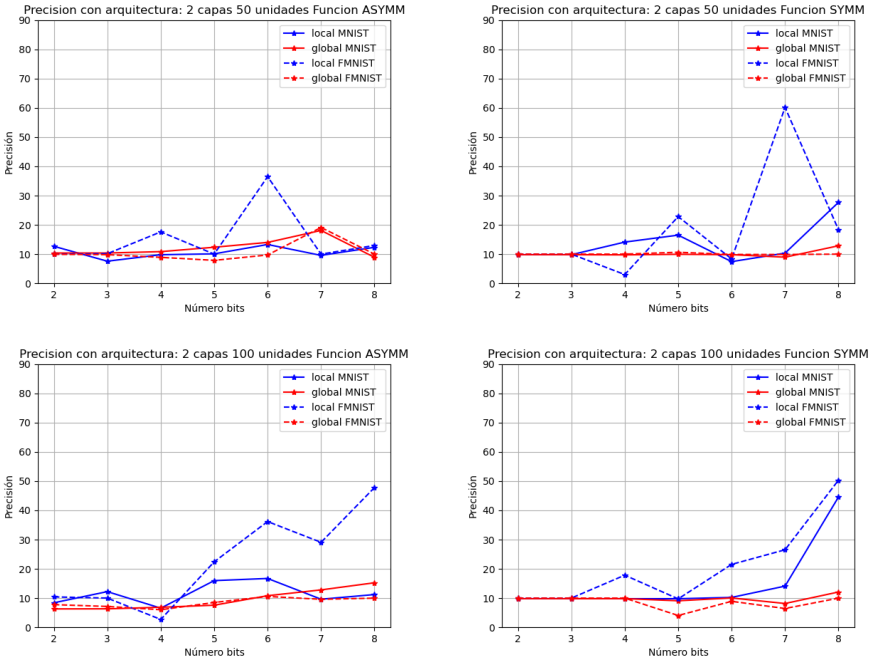


Figura 5.16: Arquitectura: 2 capas ocultas

Con los modelos de 2 capas y 50 y 100 neuronas por capa oculta, podemos ver cierta mejora con respecto a la arquitectura anterior. Sin embargo, se siguen sin conseguir resultados decentes. Lo máximo que se alcanza es un

60 % en FMNIST con el modelo cuantificado con 7 bits localmente mediante SYMM. Se observa cierta mejora cuando se aumenta la anchura de la red, pero aun así, los resultados demuestran que en arquitecturas de este tamaño el algoritmo es inviable. Veamos por último, que ocurre si aumentamos en 1 capa la profundidad de la red.

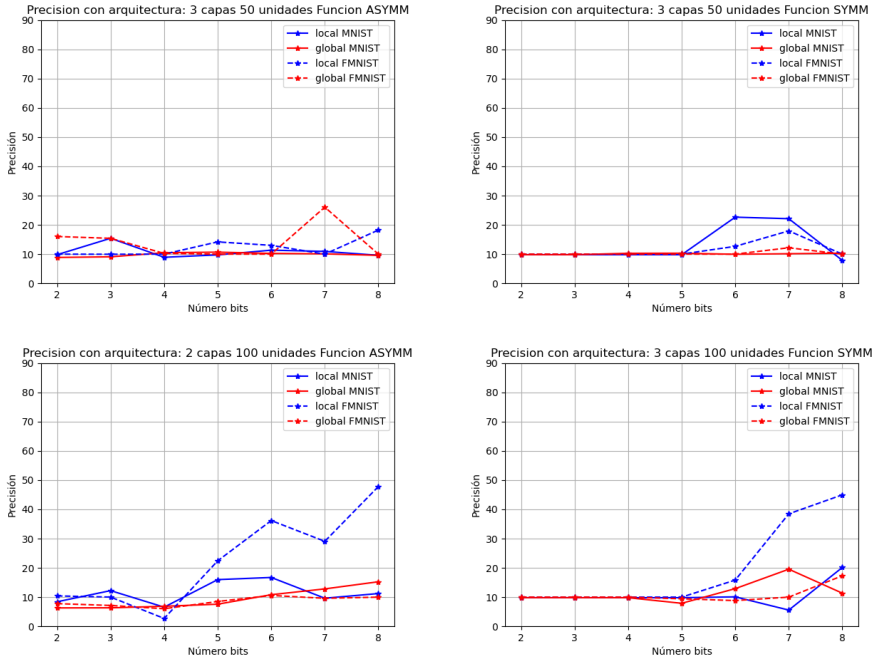
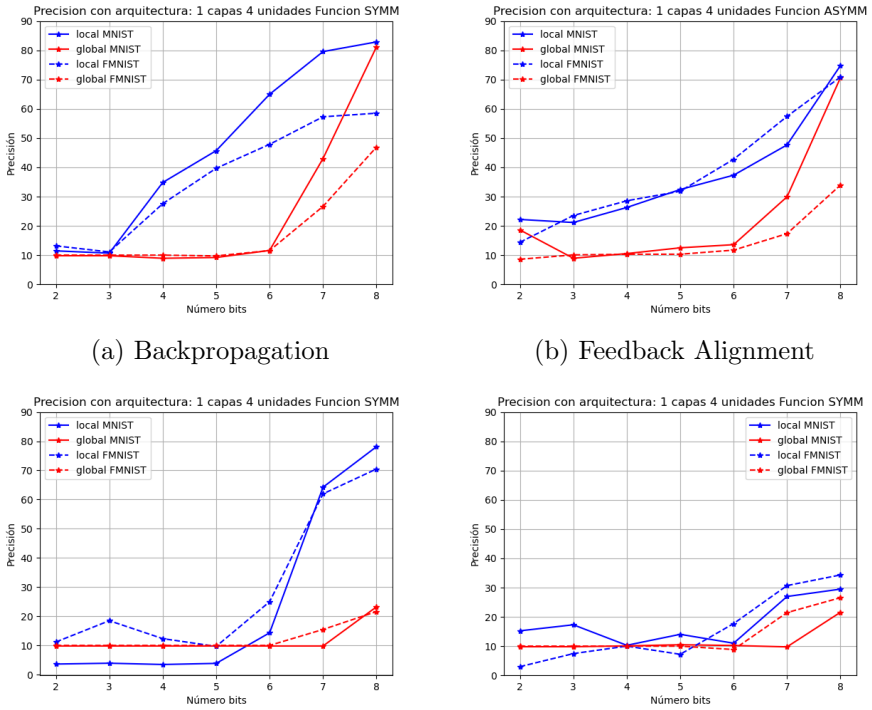


Figura 5.17: Arquitectura: 3 capas ocultas

Una vez más, aumentada la profundidad, los resultados empeoran. El aumentar la anchura tampoco permite al algoritmo alcanzar mejores óptimos. Existe cierta mejoría, pero siguen siendo precisiones muy bajas. Por lo tanto, vemos que este algoritmo no es útil para redes en circuitos neuromórficos.

5.5. Comparación de los algoritmos

Comenzaremos analizando los modelos bases.



(a) Backpropagation

(b) Feedback Alignment

(c) HSIC

(d) Synthetic Gradient

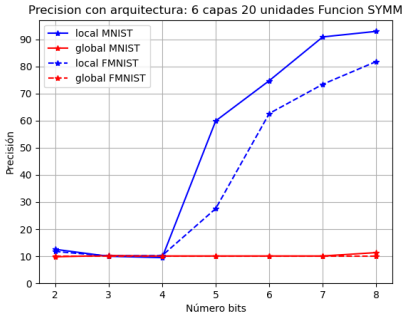
Figura 5.18: Comparación modelos bases

Comparando las 4 gráficas, el modelo que peor desempeño tiene es el de Synthetic gradient. Como se comentó en su respectivo Apartado 5.4, este algoritmo no puede ser aplicado con cuantificación (en las arquitecturas estudiadas). Como en ninguna de las arquitecturas estudiadas ha conseguido resultados viables, no se va a usar en las siguientes comparaciones.

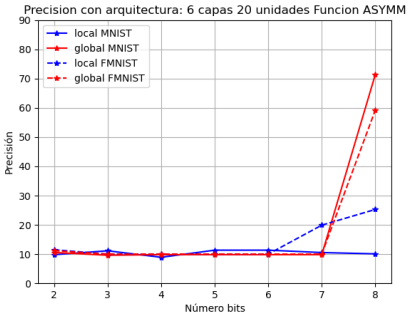
Fijémonos ahora en los 3 algoritmos restantes. Las gráficas que más se asemejan son la de Backpropagation y Feedback Alignment. Ambos algoritmos consiguen precisiones muy parecidas, sin embargo, el backpropagation obtiene mejores resultados en FMNIST cuando usa la cuantificación local. Además, en los modelos cuantificados globalmente con 7 y 8 bits, Backpropagtation es ligeramente superior. Si observamos ahora HSIC, vemos que a nivel global es inferior los dos algoritmos mencionados, y a nivel local es inferior hasta los 8 bits. Cuando se alcanza el máximo número de bits con cuantificación local, HSIC es el algoritmo que mejores precisiones alcanza, 70 % en FMNIST y 80 % en MNIST. Por lo tanto, con arquitecturas pe-

queñas y por debajo de los 8 bits, el algoritmo que mejores resultados ofrece es el clásico Backpropagation. Sin embargo, llegado a los 8 bits, el que mejor resultados alcanza es HSIC.

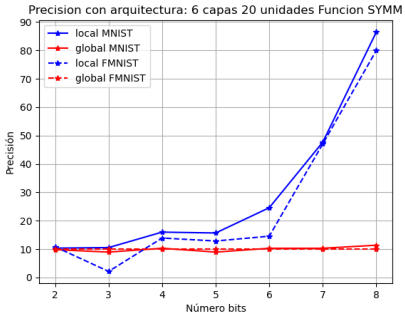
Pasemos ahora a las arquitecturas con 6 capas ocultas.



(a) Backpropagation



(b) Feedback Alignment

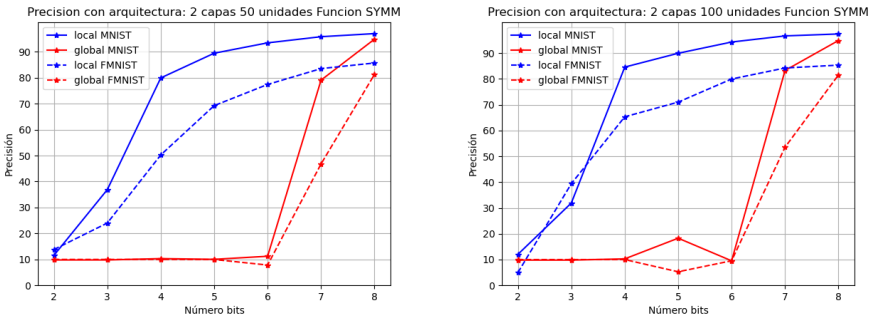


(c) HSIC

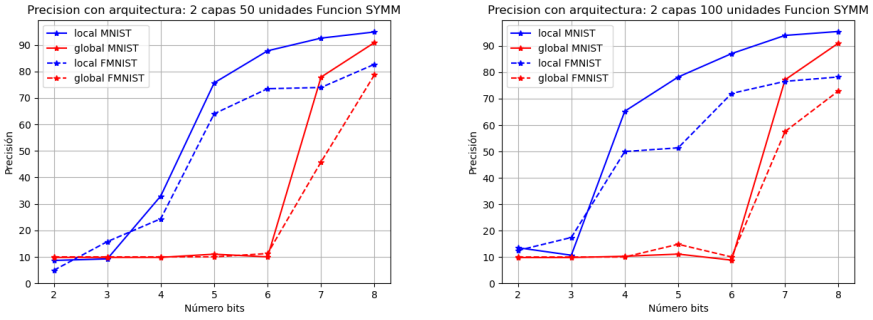
Figura 5.19: Comparación arquitecturas: 6 capas ocultas y 20 neuronas

Con cuantificación local el mejor algoritmo es el Backpropagation. 7 bits son suficientes para conseguir precisiones elevadas, 90 % MNIST y 80 % FMNIST. Mientras que para la cuantificación a nivel global, el mejor algoritmo es el de Feedback Alignment, consiguiendo un 60 % y 70 % para FMNIST y MNIST respectivamente. Por lo tanto, para modelos profundos el mejor algoritmo es el Backpropagation.

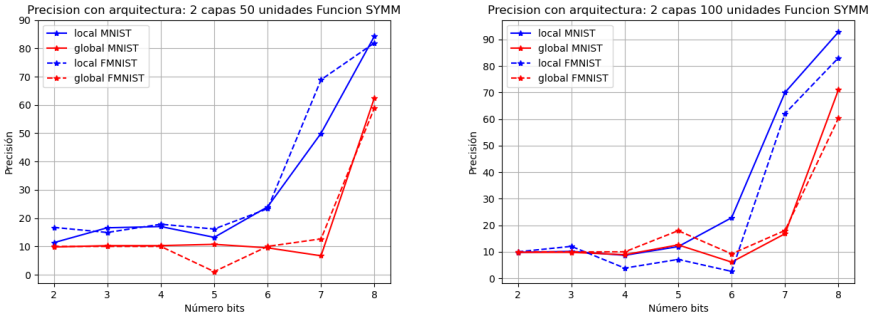
Veamos ahora los modelos con arquitecturas anchas (50-100 neuronas) y 2 capas ocultas.



(a) Backpropagation (b) Backpropagation



(c) Feedback Alignment (d) Feedback Alignment



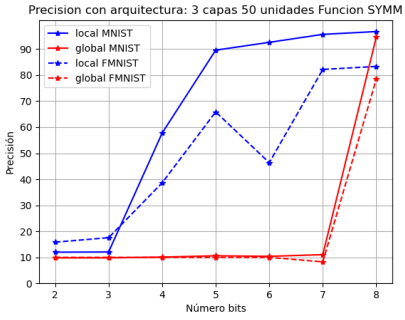
(e) HSIC (f) HSIC

Figura 5.20: Comparación arquitecturas: 2 capas ocultas y 50 y 100 neuronas

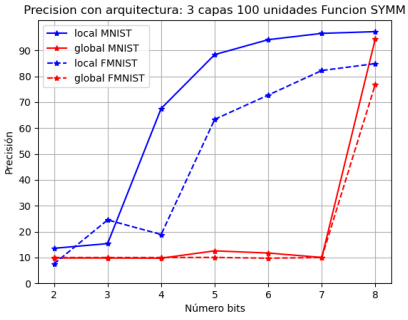
Con estas arquitecturas el algoritmo que mayor precisión obtiene con cuantificación local, es el Backpropagation. No solamente alcanza las precisiones máximas con 8 bits, si no que también es capaz de obtener precisiones elevadas a partir de 5 bits de precisión. FeedBack Alignmente le sigue muy de cerca en los resultados, aunque necesita mínimo 6 bits para conseguir precisiones elevadas. Mientras tanto, HSIC necesita 8 bits para poder competir con estos algoritmos. En cuanto a la cuantificación global, Backpropagation y Feedback Aligment están empatados, necesitando 8 bits para alcanzar

precisiones elevadas en ambos problemas. Mientras que HSIC no consigue obtener resultados tan elevados con la cuantificación global. Además de ver que algoritmo es mejor, podemos apreciar que un aumento de la anchura de la red mejora la precisión de los modelos.

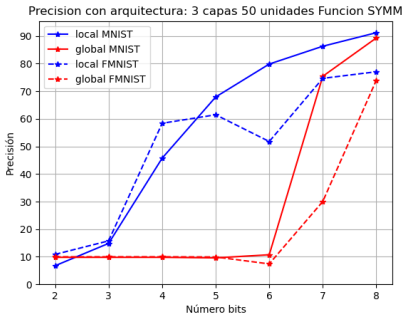
Finalmente nos queda por evaluar las arquitecturas con 3 capas ocultas y 50 y 100 neuronas.



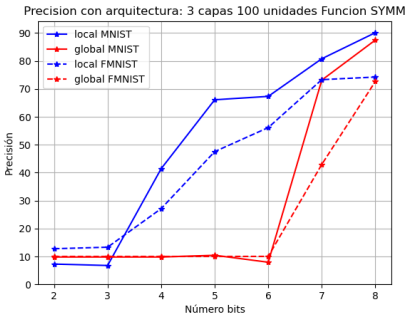
(a) Backpropagation



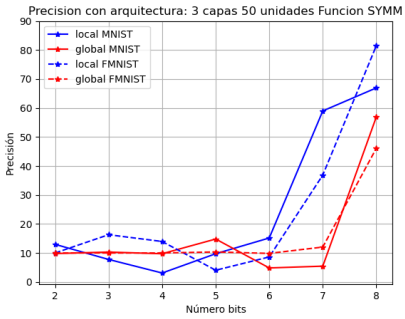
(b) Backpropagation



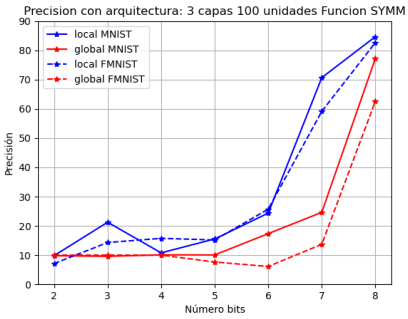
(c) Feedback Alignment



(d) Feedback Alignment



(e) HSIC



(f) HSIC

Figura 5.21: Comparación arquitecturas: 3 capas ocultas y 50 y 100 neuronas

Las gráficas son muy similares a las de las arquitecturas de 2 capas. Se

mantiene el hecho de a mayor anchura mejores resultados, y que el Back-propagation es el mejor de los 3 algoritmos. Sin embargo, en comparación con los modelos de 2 capas, las precisiones decaen ligeramente.

Capítulo 6

Conclusión y trabajo futuro

En este capítulo final se muestran las conclusiones en base a los resultados mostrados. Además, se valorará la viabilidad de aplicar estos algoritmos en circuitos neuromórficos. Para concluir, se mostrará posibles lineas de investigación a seguir en un futuro.

6.1. Revisión de las hipótesis

Expuestos los resultados de la experimentación y hechas las comparaciones entre los algoritmos, se procede a confirmar o refutar las hipótesis formuladas.

H1: Los algoritmos sin cuantificación alcanzan las mismas precisiones

Arquitectura	**BP**	**HSIC**	**FA**	**SG**
2 capas 4 unidades	87.07	77.99	75.37	38.95
6 capas 20 unidades	95.5	92.06	30.28	9.8
3 capas 100 unidades	98.0	95.22	94.98	14.17
2 capas 100 unidades	98.18	95.3	97.04	14.39
2 capas 50 unidades	97.53	95.05	96.42	56.07
3 capas 50 unidades	97.39	94.96	94.8	33.87

Cuadro 6.1: Precisiones en MNIST

Arquitectura	BP	HSIC	FA	SG
2 capas 4 unidades	81.59	71.22	77.26	47.46
6 capas 20 unidades	86.21	81.89	81.17	10.13
3 capas 100 unidades	88.6	83.65	83.74	11.34
2 capas 100 unidades	88.76	84.45	86.24	65.34
2 capas 50 unidades	88.19	84.2	85.9	68.95
3 capas 50 unidades	87.97	83.99	85.16	10.0

Cuadro 6.2: Precisiones en FMNIST

Observando los resultados en ambos problemas, podemos apreciar que las precisiones más altas son alcanzadas por el algoritmo de Backpropagation. Las precisiones alcanzadas por los algoritmos HSIC y Feedback Alignment son muy cercanas a las del Backpropagation, mientras que las alcanzadas por Synthetic Gradients son muy bajas. Por lo tanto, la hipótesis queda refutada pues no todos los algoritmos alcanzan al Backpropagation.

H2: Límite de cuantificación en torno a 6 o 7 bits

Se ha podido apreciar en la experimentación, que los algoritmos de Backpropagation y Feedback Alignment pueden conseguir precisiones elevadas por debajo de este umbral con cuantificación local. Como se ha comentado en la experimentación, este tipo de cuantificación se ha estudiado con el fin de ver si existe una mejora cuando las redes trabajan mejor con distintos rangos de valores por capa. A la luz de los resultados esto es así, por lo tanto, a futuro se tendría que investigar algún método para hacer la estimación de dichos rangos. Por su parte, centrándonos en las cuantificaciones globales (las más fieles a la realidad), podemos ver que para conseguir precisiones elevadas se necesitan 7 u 8 bits, dependiendo del problema y la arquitectura. Por lo tanto, nuestra hipótesis inicial se cumple.

H3: ASYMM y SYMM ofrecen mismo rendimiento

Con un simple vistazo a los resultados se puede apreciar que esta hipótesis queda totalmente refutada. La función ASYMM mete mucho ruido de cuantificación, llegado a los 6 bits este ruido afecta notoriamente al proceso de búsqueda, haciendo que en ciertos casos la búsqueda diverja y no encuentre óptimos. Por su parte, la función SYMM no introduce tanto ruido permitiendo a los modelos alcanzar buenos óptimos.

H4: Mejor algoritmo: HSIC

Otra hipótesis que queda refutada con los resultados. Como hemos podido apreciar, alcanzado los 8 bits este modelo consigue alcanzar precisiones elevadas, sin embargo, es superado tanto por el Backpropagation como por el Feedback Alignment. Además, con menos de 7 bits, este algoritmo ya se mediante cuantificación local o global, no consigue alcanzar precisiones elevadas.

H5: Parámetro diferencial arquitecturas: Anchura

Este hipótesis se puede confirmar con los resultados ofrecidos. En todos los algoritmos, cuando se ha aumentado la anchura de las redes, se han conseguido elevar notoriamente la precisión. Siendo mucho más importante la anchura que la profundidad. Además, hemos visto que aumentada la profundidad, la precisión puede llegar a decaer. En conclusión, las arquitecturas con anchura elevada permiten alcanzar precisiones más elevadas.

6.2. Viabilidad de las redes neuronales en circuitos neuromórficos

Ahora toca responder a la pregunta inicial de este proyecto. ¿Es viable entrenar redes neuronales totalmente conectada en circuitos neuromórficos? Con las arquitecturas estudiadas y los algoritmos seleccionados, la respuesta es: a día de hoy no es posible. La precisión alcanzada por los memristores es de 2 a 3 bits, y en base a los resultados, estas precisiones son muy pobres. Con estas arquitecturas y algoritmos estudiados hemos visto que se necesita mínimo una precisión de 7 u 8 bits para alcanzar precisiones elevadas. Por lo tanto, entrenar estos modelos en memristores de 3 bits no es posible.

6.3. Trabajo futuro

El estudio realizado durante este proyecto se ha ajustado a las horas que se emplean en un TFG. Por lo tanto, aun queda un gran número de factores por estudiar, como son:

- **Tipos de redes**: convolucionales o recurrentes, etc.
- **Hiperparámetros**: coeficiente de aprendizaje, número de iteraciones, optimizador a usar, etc.

- **Algoritmos de entrenamiento**: explorar nuevos algoritmos de entrenamiento.

Otro factor muy importante que se tendría que estudiar es el método de actualización de los pesos. Los algoritmos estudiados varían a la hora de calcular el gradiente, pero los pesos se actualizan de la misma forma: gradiente descendente. Así que estudiar una alternativa sería interesante. Por otro lado, otro tipo de red que es interesante estudiar son las redes binarias. Este tipo de redes llevan la cuantificación al extremo, usando un solo bit por peso. Este tipo de redes pueden conseguir precisiones elevadas, y debido a su cuantificación inherente, los circuitos neuromórficos son perfectos para su entrenamiento. Es por ello que continuar la investigación sobre este tipo de redes es de especial interés.

Bibliografía

[1] A. Mehonic, A. Sebastian, B. Rajendran, O. Simeone, E. Vasilaki, and A. Kenyon, "Memristors—from in-memory computing, deep learning acceleration, and spiking neural networks to the future of neuromorphic and bio-inspired computing," *Advanced Intelligent Systems*, vol. 2, 08 2020.

[2] L. Eeckhout, "Is moore's law slowing down? what's next?" *IEEE Micro*, vol. 37, no. 04, pp. 4–5, 2017.

[3] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st Conference on Computing Frontiers*, ser. CF '04. New York, NY, USA: Association for Computing Machinery, 2004, p. 162. [Online]. Available: https://doi.org/10.1145/977091.977115

[4] P. Dhar, "The carbon impact of artificial intelligence," *Nature Machine Intelligence*, vol. 2, no. 8, pp. 423–425, -08-12 2020. [Online]. Available: https://www.nature.com/articles/s42256-020-0219-9

[5] C. D. Schuman, S. R. Kulkarni, M. Parsa, J. P. Mitchell, P. Date, and B. Kay, "Opportunities for neuromorphic computing algorithms and applications," *Nature Computational Science*, vol. 2, no. 1, pp. 10–19, -01 2022. [Online]. Available: https://www.nature.com/articles/s43588-021-00184-y

[6] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: a tutorial," *Computer*, vol. 29, no. 3, pp. 31–44, March 1996.

[7] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature (London)*, vol. 323, no. 6088, pp. 533–536, Oct 9, 1986. [Online]. Available: http://dx.doi.org/10.1038/323533a0

[8] S. Grossberg, "Competitive learning: From interactive activation to adaptive resonance," *Cognitive Science*, vol. 11, no. 1, pp. 23–63, January 1, 1987. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0364021387800253

[9] E. m. Johansson, F. u. Dowla, and D. m. Goodman, "Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method," *International Journal of Neural Systems*, vol. 02, no. 04, pp. 291–301, January 1, 1991. [Online]. Available: https://www.worldscientific.com/doi/abs/10.1142/S0129065791000261

[10] H. Yu and B. M. Wilamowski, "Levenberg-marquardt training," *Industrial electronics handbook*, vol. 5, no. 12, p. 1, 2011.

[11] J. Rafati and R. F. Marcia, "Improving l-bfgs initialization for trust-region methods in deep learning," December 2018, pp. 501–508.

[12] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012. [Online]. Available: https://arxiv.org/abs/1212.5701

[13] M. Lanza, A. Sebastian, W. D. Lu, M. L. Gallo, M.-F. Chang, D. Akinwande, F. M. Puglisi, H. N. Alshareef, M. Liu, and J. B. Roldan, "Memristive technologies for data storage, computation, encryption, and radio-frequency communication," *Science*, vol. 376, no. 6597, p. eabj9979, 2022. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.abj9979

[14] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G.-J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015.

[15] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, 2014.

[16] M. Davies, N. Srinivasa, T.-H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C.-K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y.-H. Weng, A. Wild, Y. Yang, and H. Wang, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.

[17] S. GHOSH-DASTIDAR and H. ADELI, "Spiking neural networks," *International Journal of Neural Systems*, vol. 19, no. 04, pp. 295–308, 2009, pMID: 19731402. [Online]. Available: https://doi.org/10.1142/S0129065709002002

[18] R. C. Romero Zaliz, A. Cantudo, F. Jiménez Molinos, and J. B. Roldán Aranda, "An analysis on the architecture and the size of

quantized hardware neural networks based on memristors," German Research Foundation (DFG) under Project 434434223-SFB1461, 12 2021. [Online]. Available: http://hdl.handle.net/10481/72221

[19] Y. Zhang, M. Cui, L. Shen, and Z. Zeng, "Memristive quantized neural networks: A novel approach to accelerate deep learning on-chip," *IEEE Transactions on Cybernetics*, vol. 51, no. 4, pp. 1875–1887, 2021.

[20] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, "A survey of neuromorphic computing and neural networks in hardware," *CoRR*, vol. abs/1705.06963, 2017. [Online]. Available: http://arxiv.org/abs/1705.06963

[21] L. Chua, "Memristor-the missing circuit element," *IEEE Transactions on Circuit Theory*, vol. 18, no. 5, pp. 507–519, 1971.

[22] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *Nature*, vol. 453, no. 7191, pp. 80–83, -05 2008. [Online]. Available: https://www.nature.com/articles/nature06932

[23] Y. Guo, "A survey on methods and theories of quantized neural networks," 2018. [Online]. Available: https://arxiv.org/abs/1808.04752

[24] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'15. Cambridge, MA, USA: MIT Press, 2015, p. 3123–3131.

[25] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.

[26] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1510.00149

[27] Y. Choi, M. El-Khamy, and J. Lee, "Towards the limit of network quantization," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=rJ8uNptgl

[28] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *CoRR*, vol. abs/1702.03044, 2017. [Online]. Available: http://arxiv.org/abs/1702.03044

[29] W.-D. K. Ma, J. Lewis, and W. B. Kleijn, "The hsic bottleneck: Deep learning without back-propagation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, 2020, pp. 5085–5092.

[30] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, "Random synaptic feedback weights support error backpropagation for deep learning," *Nature Communications*, vol. 7, no. 1, pp. 1–10, -11-08 2016. [Online]. Available: https://www.nature.com/articles/ncomms13276

[31] M. Jaderberg, W. M. Czarnecki, S. Osindero, O. Vinyals, A. Graves, D. Silver, and K. Kavukcuoglu, "Decoupled neural interfaces using synthetic gradients," in *International conference on machine learning*. PMLR, 2017, pp. 1627–1635.

[32] M. Thomas and A. T. Joy, *Elements of information theory*. Wiley-Interscience, 2006.

[33] E. Bisong, "Benchmarking decoupled neural interfaces with synthetic gradients," December 22, 2017.

[34] P. Nayak, D. Zhang, and S. Chai, "Bit efficient quantization for deep neural networks," 2019.

[35] J. Sutherland, *Jeff Sutherland's Scrum Handbook*, 01 2010.

[36] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: Extending mnist to handwritten letters," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017, pp. 2921–2926.

[37] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *CoRR*, vol. abs/1708.07747, 2017. [Online]. Available: http://arxiv.org/abs/1708.07747

[38] K. Dinghofer and F. Hartung, "Analysis of criteria for the selection of machine learning frameworks," in *2020 International Conference on Computing, Networking and Communications (ICNC)*, 2020, pp. 373–377.

[39] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, "Backpropagation and the brain," *Nature Reviews Neuroscience*, vol. 21, no. 6, pp. 335–346, -06 2020. [Online]. Available: https://www.nature.com/articles/s41583-020-0277-3