



Descomposiciones Tensoriales Aplicadas a Arquitecturas de Deep Learning

Tesis de Grado en Ingeniería Informática

Facultad de Ingeniería
Universidad de Buenos Aires

Director: Dr. Ing. Cesar F. CAIAFA
ccaiafa@fi.uba.ar

Autor:
Andres OTERO
anotero@fi.uba.ar

June 21, 2021

Resumen

Las redes neuronales profundas han mostrado recientemente muy buenos resultados, realizando diferentes tareas de aprendizaje de manera muy efectiva. Las descomposiciones tensoriales permiten reducir el espacio de memoria y las operaciones para el procesamiento de estructuras de datos multidimensionales. Varios investigadores han logrado explotar estas propiedades para comprimir arquitecturas de *deep learning*, además de proveer un marco teórico para interpretar éstas desde su punto de vista. Este trabajo plantea explorar una variante, utilizar una arquitectura basada en la descomposición *Tensor Ring*, se hicieron pruebas con diferentes bases de datos para en primera instancia validar su funcionamiento y luego comprobar su eficacia. Se alcanzaron resultados para considerarla efectiva pero con resultados que no son comparables con el estado del arte actual, aun así abre el espacio para nuevas variaciones y otros experimentos con esta nueva arquitectura.

Agradecimientos

A la *Facultad de Ingeniería de la Universidad de Buenos Aires* por la formación que me brindo.

Al *Dr. Cesar Caiafa*, director de esta tesis por haber guiado con su sabiduría y experiencia esta tesis de Ingeniería Informática.

A mi familia por haberme acompañado durante toda la carrera y durante la elaboración de este trabajo.

A mis amigos por compartir todo este camino conmigo, apoyándome en los buenos y los malos momentos.

ÍNDICE GENERAL

Resumen	I
Agradecimientos	II
Índice general	IV
Lista de figuras	V
Lista de tablas	VII
1. Capítulo 1: Introducción	1
1.1. Contribuciones de esta tesis	3
1.2. ¿Por qué utilizar descomposiciones tensoriales en <i>Deep Learning</i> ?	3
1.3. Desarrollo de la tesis	5
2. Capítulo 2: Redes Neuronales Profundas	6
2.1. Introducción	7
2.2. Redes Neuronales Convolucionales	9
2.3. Redes Neuronales Recurrentes	11
2.4. Vanilla RNN	12
3. Capítulo 3: Descomposiciones tensoriales	16
3.1. Notación de Tensores	18
3.1.1. Notación de Diagrama de Tensores	18
3.2. Historia	20
3.3. Tensor Train	22
3.4. Tensor Ring	23
4. Capítulo 4: El rol de las descomposiciones tensoriales en deep learning	26
4.1. Compresibilidad de redes neuronales profundas	27
4.2. Expresividad	29
4.3. Arquitectura propuesta: Tensor Ring en deep learning	31
4.3.1. La nueva arquitectura es compatible con una RNN	34
4.3.2. Cómo aplicar el modelo a un conjunto de entradas (<i>batch</i>)	36
4.3.3. Arquitecturas paralela, serial y compartida	37
4.4. Comparación con arquitecturas anteriores	40
4.4.1. Tensor Train (TT)	40
5. Capítulo 5: Resultados Experimentales	48
5.1. Bases de datos utilizadas	49
5.1.1. MNIST y Fashion-MNIST	49
5.1.2. CIFAR-10	53
5.1.3. IMDB	55
5.2. Arquitecturas de referencia	59
5.2.1. MNIST y Fashion-MNIST	59
5.3. CIFAR 10	59
5.3.1. IMDB	63

5.4.	Detalles de implementación y selección de hiperparámetros	65
5.5.	Resultados	66
5.5.1.	MNIST	66
5.5.2.	Fashion-MNIST	70
5.5.3.	CIFAR-10	74
5.5.4.	IMDB	76
6.	Capítulo 6: Conclusiones	80
6.1.	Conclusiones Generales	81
6.2.	Conclusiones Experimentales	81
6.3.	Futuras líneas de investigación	83
A.	Apéndice	93
A.1.	Demostraciones y ecuaciones	93
A.2.	LSTM	94
A.3.	GRU	95

LISTA DE FIGURAS

1.	Precisión, operaciones y cantidad de parámetros de diferentes redes	2
2.	Alexnet	4
3.	Diagrama MLP	8
4.	CNN interacciones dispersas	10
5.	CNN compartir parámetros	10
6.	Diagrama RNN	12
7.	Diagrama RNN con una sola salida	12
8.	Diagrama Celda RNN	13
9.	Diagrama RNN <i>Elman network</i>	14
10.	Diagrama RNN <i>Jordan network</i>	14
11.	Ejemplo <i>fiber</i> en un tensor	17
12.	Ejemplo <i>slice</i> en un tensor	17
13.	Ejemplos notación de diagrama de tensores	19
14.	Ejemplos notación de diferentes contracciones	20
15.	Ejemplo descomposición CPD	21
16.	Ejemplo descomposición de Tucker	22
17.	Ejemplo descomposición Jerárquica de Tucker	22
18.	Ejemplo descomposición <i>Tensor Train</i>	23
19.	Ejemplo descomposición Tensor Ring	25
20.	Ejemplo Tensor Feature	33
21.	Ejemplo <i>score function</i>	33
22.	Ejemplo Tensor Red	34
23.	Ejemplo <i>score function</i> con una descomposición <i>Tensor Ring</i>	36
24.	Ejemplo <i>score function</i> con una descomposición <i>Tensor Ring</i> con una dimensión de <i>batch</i>	37
25.	Arquitectura <i>Tensor Ring</i> paralela	38
26.	Arquitectura <i>Tensor Ring</i> serial	39
27.	Arquitectura Tensor Ring <i>compartida</i>	39
28.	Ejemplo <i>score function</i> con una descomposición <i>Tensor Train</i>	41
29.	Ejemplo <i>score function</i> con una descomposición <i>Tensor Train</i>	43
30.	Arquitectura <i>Tensor Train</i> paralela	44
31.	Arquitectura <i>Tensor Train</i> serial	44
32.	Arquitectura <i>Tensor Train</i> compartida	45
33.	Ejemplo MNIST	49
34.	Ejemplo <i>Fashion-MNIST</i>	50
35.	Ejemplo de aplanamiento de divisiones de una muestra de <i>MNIST</i>	51
36.	Ejemplo varias muestras de <i>MNIST</i> aplanadas	52
37.	Ejemplo varias muestras de <i>Fashion MNIST</i> aplanadas	52
38.	Ejemplo <i>CIFAR-10</i>	53
39.	Ejemplo de una muestra de <i>CIFAR-10</i> normalizada y aumentada	54
40.	Tokenizer	56
41.	GloVe	57
42.	Distribución <i>IMDB</i>	58
43.	Diagrama de la CNN de referencia	60
44.	Diagrama del bloque de la capa convolucional	62
45.	Red convolucional integrada con la arquitectura de tensores.	63

46.	Convolución texto	64
47.	Interpolación de las mejores resultados sobre el conjunto de los test respecto del rango para <i>MNIST</i> con $m = 32$	67
48.	Interpolación de las mejores resultados sobre el conjunto de los test respecto del rango de <i>MNIST</i> con $m = 64$	67
49.	Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros de <i>MNIST</i> con $m = 32$	67
50.	Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros de <i>MNIST</i> con $m = 64$	67
51.	Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros para cada una de las redes con <i>MNIST</i>	68
52.	Interpolación de las mejores resultados sobre el conjunto de los test respecto del rango de <i>Fashion-MNIST</i> con $m = 32$	71
53.	Interpolación de las mejores resultados sobre el conjunto de los test respecto del rango de <i>Fashion-MNIST</i> con $m = 64$	71
54.	Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros de <i>Fashion-MNIST</i> con $m = 32$	71
55.	Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros de <i>Fashion-MNIST</i> con $m = 64$	71
57.	Parámetros para los ensayos para <i>MNIST</i> y <i>Fashion-MNIST</i>	73
58.	Diagrama de una celda LSTM	95
59.	Diagrama de una celda GRU	96

LISTA DE TABLAS

1.	Notación básica de Tensores	18
2.	Notación de operaciones de Tensores	19
3.	Correspondencia entre descomposición y redes de deep learning [52].	31
4.	Comparación del poder expresivo de varias redes [52]. Dado un red de ancho r , especificada en una columna, la fila corresponde al límite superior de la profundidad de una red equivalente de otro tipo.	31
5.	Estado del arte de las bases de datos utilizadas	53
6.	Mejor Rendimiento de las diferentes redes con arquitectura <i>compartida</i> para <i>MNIST</i> con $m=64$	68
7.	Mejor Rendimiento de las diferentes redes con arquitectura <i>compartida</i> para <i>MNIST</i> con $m=32$	69
8.	Mejor Rendimiento de las diferentes redes con arquitectura <i>compartida</i> para <i>Fashion-MNIST</i> con $m=64$	72
9.	Mejor Rendimiento de las diferentes redes con arquitectura <i>compartida</i> para <i>Fashion-MNIST</i> con $m=32$	73
10.	Mejor Rendimiento de las diferentes redes para <i>CIFAR-10</i>	75
11.	Mejor Rendimiento de las diferentes redes para la base de datos <i>IMDB</i>	77

CAPÍTULO 1: INTRODUCCIÓN

En este capítulo, se introducen los principales temas a tratar en el trabajo de tesis. Se plantean los principales conceptos, citando las referencias más importantes y explicando la motivación detrás del nuevo desarrollo propuesto. Además, se describen los principales resultados y contribuciones que aporta este trabajo. Finalmente, se detallan de cuáles son los temas a tratar en cada uno de los capítulos.

Las redes neuronales profundas, también referidas como arquitecturas de *Deep Learning* [32], han demostrado tener la capacidad de realizar tareas de aprendizaje de manera muy efectiva. Esta tecnología se aplica a innumerables problemas de inteligencia artificial que incluyen, por ejemplo, el desarrollo de vehículos autónomos [5], diagnóstico de enfermedades [26], asistentes activados por voz (Siri, Alexa, etc.), traducción automática entre idiomas [1, 69], predicción de terremotos [23] y muchos más.

Si bien las redes neuronales habían sido propuestas y utilizadas para reconocimiento de patrones hace algunas décadas [4, 27, 64], sólo recientemente éstas han logrado revolucionar el estado del arte en inteligencia artificial debido principalmente a dos factores: (1) la disponibilidad de grandes volúmenes de datos (*big data*) que permiten entrenar arquitecturas de cientos de miles de millones de parámetros, y (2) el avance tecnológico y abaratamiento del hardware para cálculo intensivo a través de arquitecturas paralelas implementadas en circuitos integrados para procesamiento gráfico (*Graphic Processor Unit* - *GPU*).

La versatilidad de las arquitecturas de *deep learning* conlleva una serie de problemas que representan el desafío de las investigaciones actuales. El entrenamiento de estos modelos requiere ajustar una cantidad inmensa de parámetros, lo que en la práctica exige grandes volúmenes de almacenamiento y potencia de cómputo, como se ilustra en la Figura 1 para el caso de clasificación automática de imágenes naturales.

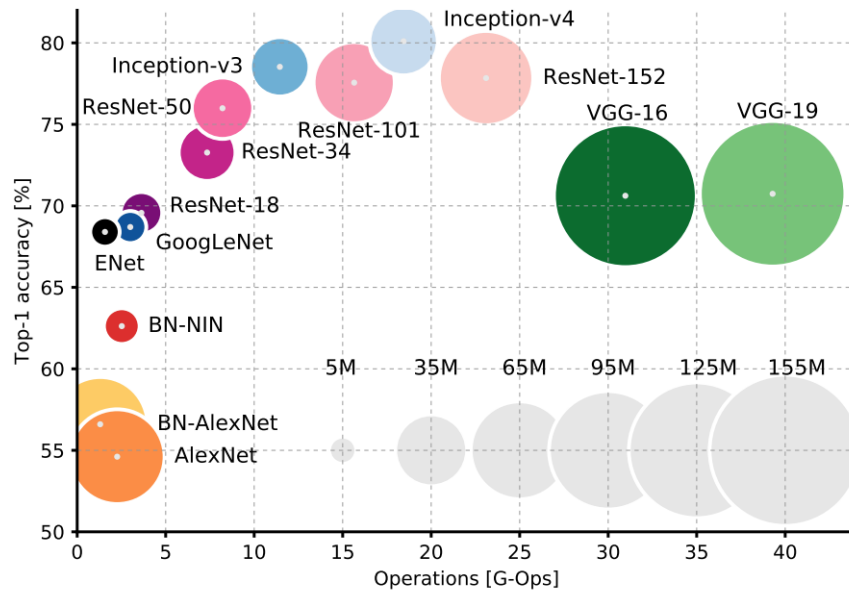


Figura 1: (figura extraída de [7]) Comparación de precisión (eje y), número de operaciones (eje x) y cantidad de parámetros (radio de círculos), para varias arquitecturas de *deep learning* de la competición *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*, 2016. Es notable que todas estas arquitecturas tienen varios millones de parámetros.

Por otro lado, durante los últimos veinte años aproximadamente, han habido grandes avances en el estudio de técnicas de procesamiento de señales basadas en descomposiciones tensoriales [15, 84]. Un tensor, o arreglo multidimensional, es un objeto matemático que representa la generalización

de un arreglo unidimensional (vector) y bidimensional (matriz) al caso multidimensional. Una descomposición tensorial permite representar un tensor, que potencialmente es grande o contiene muchas dimensiones, a través del producto de un conjunto de tensores de menor tamaño y número de dimensiones. Estas descomposiciones son atractivas ya que permiten reducir el número de operaciones y requerimiento de memoria a la hora de implementar métodos de procesamiento con datos multidimensionales permitiendo así combatir el efecto de la maldición de la dimensionalidad (*curse of dimensionality*) [2]. Las descomposiciones tensoriales propuestas en la literatura tienen distintas formas y propiedades. Entre las más populares podemos mencionar: *Canonical / PARAFAC decomposition* (CPD) [9, 36, 53], la descomposición de Tucker [92] y las Redes Tensoriales (*Tensor Networks*) entre las cuales se encuentran *Tensor Train* (TT) [75], *Tensor Ring* (TR) [98] y Hierarchical Tucker decomposition (HT) [35].

Recientemente, varios investigadores han propuesto utilizar descomposiciones tensoriales en las arquitecturas de *deep learning*. Al mismo tiempo, las redes tensoriales permiten una nueva interpretación de las redes neuronales profundas [14, 16, 18, 28, 52, 74]. El objetivo principal de este trabajo de tesis es analizar resultados recientes que vinculan las descomposiciones tensoriales con arquitecturas de *deep learning*.

1.1. Contribuciones de esta tesis

La principal contribución de esta tesis es un análisis comparativo de los rendimientos obtenidos aplicando las descomposiciones TT y TR como base de la arquitectura de redes neuronales para problemas de clasificación supervisada de imágenes y otros tipos de datos. Los principales resultados obtenidos en esta tesis son:

- Se encontraron rendimientos similares en la precisión de clasificación para las redes TT y TR de similar cantidad de parametros en diferentes bases de datos como *CIFAR-10* y *MNIST*.
- La red TT demostró rendimientos superiores sobre la red TR en la precisión de clasificación para redes de similar cantidad de parametros en las bases de datos de *Fashion-MNIST* y *IMDB*.
- La red TR, con valores similares de rango y tamaño de *feature map*, resulta menos eficiente tanto en cantidad de operaciones como en el tiempo de clasificación que la red TT. Lo mismo sucede cuando la cantidad de parámetros de ambas redes son comparables.

Las conclusiones más importantes son:

- La red TR demostró ser capaz de clasificar las diferentes bases de datos utilizadas en el trabajo, siendo éstas diversas en su naturaleza y con distintos niveles de complejidad, inclusive consiguiendo rendimientos similares a la red TT en algunos casos.
- La red TT debido a los resultados obtenidos y teniendo en cuenta otras variables como los tiempos de clasificación y el número de parámetros, demostró ser una opción superadora a la red TR.

1.2. ¿Por qué utilizar descomposiciones tensoriales en *Deep Learning*?

La popularidad del *deep learning* no se ha detenido desde su inicio con la tercera ola de la inteligencia artificial, alrededor de 2006 [39] cuando, ayudado por el aumento exponencial de la capacidad de procesamiento, se empezaron a utilizar modelos cada vez más profundos, es decir, redes con número de capas neuronales cada vez más grandes. Además, el volumen de los datos se

1.3. Desarrollo de la tesis

Este trabajo está dividido en 6 capítulos. En este primer capítulo se da una breve introducción a los temas desarrollados, además de presentar un corto resumen de los principales resultados y conclusiones. En el próximo capítulo se introducen diferentes redes neuronales que luego serán utilizadas como referencia y darán contexto a todo lo relacionado a éstas para el resto de la tesis. En el tercer capítulo se da una recopilación similar para las descomposiciones tensoriales, dando un marco histórico de éstas y ahondando en las descomposiciones más relevantes para este trabajo.

En el cuarto capítulo se explica el rol de las descomposiciones tensoriales en *deep learning*, mostrando las diferentes maneras en las cuales son utilizadas en esta área. En este capítulo también se incluye el desarrollo de la arquitectura propuesta *Tensor Ring* (TR) y los detalles de implementación para diferentes bases de datos, tanto de ésta como de la arquitectura *Tensor Train* (TT) que servirá como punto de comparación. En el quinto capítulo se muestran los resultados obtenidos para la arquitectura propuesta y las otras redes utilizadas como referencia para las diferentes bases de datos. En el último capítulo se exponen de manera más profunda las conclusiones desarrolladas en base a los resultados encontrados, además se plantean nuevas líneas de investigación que surgen a partir de lo desarrollado en este trabajo.

CAPÍTULO 2: REDES NEURONALES PROFUNDAS

Este capítulo intenta dar una pequeña introducción al campo de las redes neuronales profundas. Éstas se han vuelto muy populares en los últimos años, obteniendo muy buenos resultados en diferentes aplicaciones con una gran variedad de usos en disciplinas varias. Se hace un repaso detallando las redes que serán utilizadas posteriormente en el trabajo.

2.1. Introducción

Las redes neuronales profundas *feedforward* [32] también conocidas como *multilayer perceptrons* (MLP) son uno de los modelos más conocidos de *deep learning*. Su objetivo principal es el de aproximar una función f^* , por ejemplo un clasificador, la función $y = f^*(\mathbf{x})$ asigna la categoría y a la entrada $\mathbf{x} \in \mathbb{R}^N$. Los modelos *feedforward* contienen una concatenación de capas neuronales, sin realimentación, es decir, la información fluye en un solo sentido desde la entrada hacia la salida.

Cada capa en una red neuronal representa una operación lineal sobre su entrada, seguida por una función no-lineal. La cantidad de capas determina la profundidad del modelo, la última capa es llamada la capa de salida mientras que las intermedias son capas ocultas. La operación realizada por las capas ocultas de una MLP puede escribirse como:

$$\mathbf{h}^{(t)} = \theta^{(t)} \left(\mathbf{W}^{(t)} \mathbf{x}^{(t-1)} + \mathbf{b}^{(t)} \right) \quad (\text{Ec. 2.1})$$

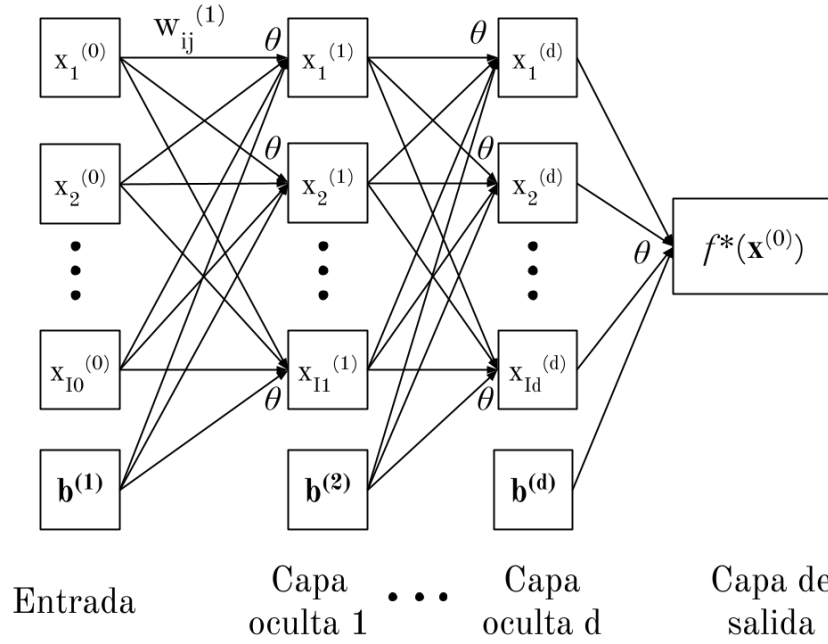
Siendo $\mathbf{h}^{(t)} \in \mathbb{R}^{I_t}$ el estado oculto de la capa t , $\mathbf{x}^{(0)} \in \mathbb{R}^{I_0}$ la entrada de la red. La matriz $\mathbf{W}^{(t)} \in \mathbb{R}^{I_t \times I_{t+1}}$ y el vector de *bias* $\mathbf{b}^{(t)} \in \mathbb{R}^{I_{t+1}}$ constituyen los parámetros de la capa y $\theta^{(t)}$ es una función no-lineal conocida como función de activación. Entre las funciones de activación más comúnmente utilizadas se encuentran: *Rectifier Linear Unit* (ReLU) [73], la función sigmoide y la tangente hiperbólica que se definen a continuación:

$$\text{ReLU } \theta(z) = \max(z, 0) \quad (\text{Ec. 2.2})$$

$$\text{Sigmoide } \theta(z) = \frac{1}{1 + e^{-z}} \quad (\text{Ec. 2.3})$$

$$\text{Tanh } \theta(z) = \frac{e^{2z} - 1}{e^{2z} + 1} \quad (\text{Ec. 2.4})$$

Estas redes son llamadas *neuronales* porque están inspiradas en las redes biológicas estudiadas por las Neurociencias [71]. Cada capa oculta consiste en un conjunto de neuronas. Cada neurona recibe como entrada las salidas de las neuronas de la capa anterior, multiplicada por pesos sinápticos. Así, cada neurona combina todas sus entradas aplicando la función de activación (Figura 3).


 Figura 3: Una red MLP [32] de profundidad d que procesa la entrada \mathbf{x} y tiene salida y .

Muchas veces a la salida de una red neuronal, se suele utilizar la función *softmax* [6]. Esto convierte la salida $y' \in \mathbb{R}^k$ de la última capa en una distribución de probabilidades, una para cada categoría y_k a clasificar:

$$p(y_k | \mathbf{x}^{(0)}) = \frac{e^{y'_k}}{\sum_{i=1}^K e^{y'_i}} \quad \forall k \in K \quad (\text{Ec. 2.5})$$

Otra ecuación utilizada es la llamada *logsoftmax* que es aplicar a la función *softmax* un logaritmo, la razón por la cual se utiliza es porque es numéricamente más estable (Ec. A.3) (ver apéndice).

Uno de los elementos clave a la hora de definir cualquier algoritmo de *machine learning* es la función de costo. Ésta es la encargada de definir la eficacia del algoritmo implementado, cuantifica el error entre el valor esperado y el valor de salida, permitiendo así ajustar los pesos sinápticos durante el entrenamiento de la red. Una de las funciones utilizadas para este fin es la función *negative log-likelihood*, que utiliza el principio de la estimación de máxima verosimilitud de tomar como estimación de un parámetro de un evento probabilístico, aquel que da más probabilidad de que ocurra dicho evento. Dado un conjunto de N muestras con un valor esperado $\mathbf{y}^i \in \mathbb{R}^K$ para cada muestra para K categorías, el algoritmo tiene una salida $\mathbf{y}^{i'}$ en \mathbb{R}^K , la función de costo para todo el conjunto queda definida como:

$$\mathcal{L} = - \sum_i^N \sum_k^K y_k^i \ln y_k^{i'} \quad (\text{Ec. 2.6})$$

El objetivo final de la etapa de entrenamiento es reducir al mínimo el resultado de la función costo, de esa manera consiguiendo que el algoritmo se parezca lo más posible a la función original que intenta emular. Para poder lograr esto, se utilizan métodos de optimización, uno de los más utilizados, es la técnica llamada *gradient descent* [11], basado en la idea de proveer actualizaciones

en dirección opuesta al gradiente de manera que la función es minimizada en cada iteración. Para las redes neuronales *feedforward*, el algoritmo denominado *back-propagation*, es el que permite que la información del costo fluya hacia atrás a través de la red, en orden para computar el gradiente. Suele confundirse como que éste es todo el algoritmo de aprendizaje para las redes neuronales *feedforward*, éste solo sirve para computar el gradiente, mientras que otros algoritmos como *Stochastic Gradient Descent* (SGD) [81] o Adam [56], son los encargados de realizar el aprendizaje usando este gradiente.

2.2. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (*Convolutional Neural Networks* o CNN) son un tipo especializado de red neuronal para procesar datos con una topología similar a una cuadrícula. Fueron desarrolladas para procesar imágenes imitando el comportamiento de las redes neuronales biológicas encontradas en los mamíferos. Sin embargo, este tipo de redes tienen muchos ejemplos de aplicaciones exitosas en una gran variedad de campos, no solo en imágenes. A diferencia de una capa completamente conectada, el nombre indica que la red emplea en algunas de sus capas una operación matemática llamada convolución. Ésta es un tipo especial de operación lineal, es un operador matemático que transforma dos funciones x y w , en una tercera función s , se puede denotar con un asterisco de la siguiente manera:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \quad (\text{Ec. 2.7})$$

En el caso de una red neuronal, el primer argumento (la función x) es considerado la entrada de la red, y el segundo argumento (la función w) sería el núcleo o *kernel*, y la salida es denominada como *feature map*. Generalmente la entrada de la red es un tensor de datos, por ejemplo una imagen, y el núcleo es un tensor de parámetros que suelen ser ajustados a través de un algoritmo de aprendizaje. Dado que cada elemento de la entrada y el núcleo deben almacenarse explícitamente por separado, se puede implementar la suma infinita como una suma sobre un número finito de elementos de la matriz. Teniendo en cuenta esto se puede escribir una convolución de una imagen \mathbf{I} de dos dimensiones con un núcleo \mathbf{K} de la siguiente manera:

$$\mathbf{S}(t) = (\mathbf{I} * \mathbf{K})(t) = \sum_m \sum_n i_{m,n} k_{i-m, j-n} \quad (\text{Ec. 2.8})$$

Una de las principales características de una capa convolucional es que tiene interacciones dispersas, también llamada conectividad dispersa. A diferencia del caso de una capa completamente conectada, que como su nombre indica tiene todas las entradas conectadas a través de la multiplicación de una matriz de pesos a todas las salidas, la capa convolucional tiene esos pesos dispersos. Esto es el resultado de tener un *kernel* más pequeño que la entrada y tiene el efecto de que éste puede extraer las características más importantes de las entradas, por ejemplo los bordes de una imagen, utilizando muchos menos parámetros y, por lo tanto, también muchas menos operaciones.

3 ₀	3 ₁	2 ₂	1	0
0 ₂	0 ₂	1 ₀	3	1
3 ₀	1 ₁	2 ₂	2	3
2	0	0	2	2
2	0	0	0	1

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

Figura 4: (figura extraída de [24]) Ejemplo de una convolución de dos dimensiones sobre una matriz de 5x5 con un núcleo de tamaño 3x3.

Otra característica clave de las capas convolucional es su capacidad de compartir parámetros, esto se refiere a que se usa el mismo parámetro para más de una función en el modelo. Comparándola nuevamente con una capa completamente conectada, cada elemento de la matriz de peso es usada solamente una vez. Mientras que en el caso de una capa convolucional, los pesos están atados entre sí, porque cada valor del *kernel* es usado en cada posición de la entrada. Esto también le da propiedad de ser invariante a traslaciones.

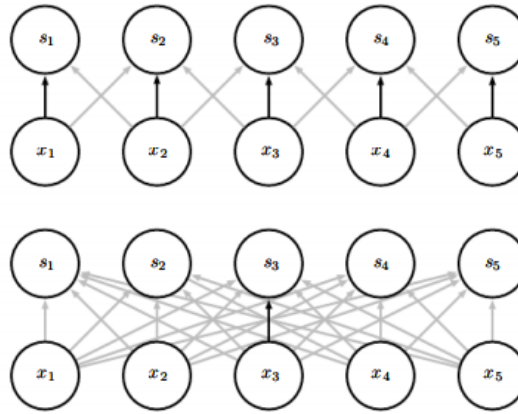


Figura 5: (figura extraída de [32]) Se representa la entrada x y la salida s de una capa convolucional de núcleo de tamaño 3 (arriba) y una capa completamente conectada (abajo). En la capa convolucional se puede ver como un mismo parámetro es compartido por todas las entradas, mientras que en la capa completamente conectada, el parámetro destacado sólo se usa en una de las entradas.

Una capa de una CNN suele estar compuesta de 3 etapas. En la primer etapa, la capa convolucional aplica la operación detallada anteriormente (Ec. 2.7). La siguiente capa suele ser una función de activación no lineal, como alguna de las funciones ya antes mencionadas, como ReLU (Ec. 2.2), Sigmoide (Ec. 2.3) o tangente hiperbólica (Ec. 2.4). La última etapa es una función de *pooling*, la cual reemplaza la salida de la etapa anterior en una ubicación determinada con una operación estadística de las salidas cercanas. Una de las operaciones es por ejemplo la función *max pooling*, que reporta la salida máxima dentro de un vecindario rectangular. Otras funciones muy utilizadas son el

promedio de un vecindario rectangular, la norma L^2 de un vecindario rectangular, o un promedio ponderado basado en un punto central.

La operación de *pooling* permite que la representación se vuelva invariante a las traslaciones pequeñas en la entrada de la capa. Si hay una pequeña traslación, los valores de salida de esta etapa no deberían variar. Esto es bueno para detectar ciertas características sin darle tanta relevancia a dónde se ubica exactamente ésta.

En la mayoría de las CNNs se suele agregar una capa completamente conectada, que ayuda combinar las características representadas por las salidas de la capa convolucional. Un buen ejemplo de una CNN y su arquitectura es la red *AlexNet* [60], que puede verse en la Figura 2.

2.3. Redes Neuronales Recurrentes

Las redes neuronales recurrentes (RNN) son una familia de redes neuronales para procesar datos secuenciales [32, 82]. Las secuencias de valores $\mathbf{X} \in \mathbb{R}^{N \times \tau}$ son de la forma $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)} \in \mathbb{R}^\tau$. Las RNN pueden también procesar secuencias de tamaños variables, por lo tanto τ puede no tener un valor fijo. El valor del índice $t = 1..\tau$ para cualquier elemento $\mathbf{x}^{(t)}$ no necesariamente indica un paso en el tiempo, sino también la posición del mismo en la secuencia. Esto implica que pueden utilizarse las RNNs no solamente para eventos temporales, sino también para elementos que pueden ser separados por posición como por ejemplo una imagen, separando pedazos de la imagen por posición en la secuencia o un vídeo haciendo el caso análogo a la imagen, pero agregándole el componente del tiempo.

La idea original que llevó de las redes MLPs a las RNNs es el de compartir parámetros a través de diferentes posiciones de las secuencias, lo cual permite extender y aplicar el modelo a diferentes ejemplos y poder generalizarlo. Si se tuvieran diferentes parámetros para cada valor de la secuencia, no se podrían generalizar para secuencias no vistas durante el entrenamiento. Tampoco permitiría compartir ese conocimiento a través de diferentes secuencias y a través de la repetición de diferentes elementos en distintas posiciones en el tiempo. Las RNNs implementan ese comportamiento de compartir parámetros utilizando recursivamente la salida anterior del modelo. Esta manera de generar resultados a través de la recurrencia produce ese comportamiento.

La manera en la cual las redes neuronales utilizan los bucles de la función para retener información está basado en la conectividad cíclica de las neuronas del cerebro [34]. La ecuación con la que se puede modelar una red RNN es similar a la de un sistema dinámico (Ec. 2.9), que tiene un estado oculto $\mathbf{h}^{(t)}$ en el tiempo t , con una entrada $\mathbf{x}^{(t)}$ para el mismo tiempo y un grupo de parámetros $\boldsymbol{\theta}$ de la función f que lo modela:

$$\mathbf{h}^{(t)} = f(\mathbf{h}^{(t-1)}, \mathbf{x}^{(t)}; \boldsymbol{\theta}) \quad (\text{Ec. 2.9})$$

Este procesamiento (Ec. 2.9) puede graficarse como un diagrama de un circuito que funciona en tiempo real, que se alimenta de una entrada y una retroalimentación por cada instante de tiempo (Figura 6). Cada nodo representa un paso en el tiempo donde, en cada momento, es calculado el estado oculto. Esto permite ver que se puede procesar la entrada sin importar el largo de la secuencia y es posible usar la misma función f para cada paso.

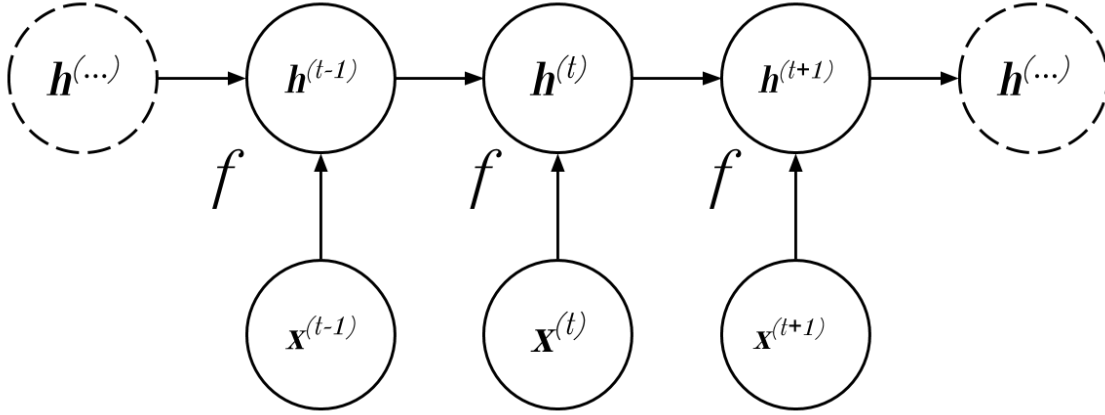


Figura 6: Una red recurrente que procesa la entrada \mathbf{x} y el estado actual \mathbf{h} a través del tiempo (Ec. 2.9).

2.4. Vanilla RNN

A la red RNN más difundida le atribuiremos el nombre de *Vanilla RNN*. La arquitectura que se referencia para todas las redes es la de una red recurrente que tiene como entrada una secuencia entera y produce una única salida (Figura 7).

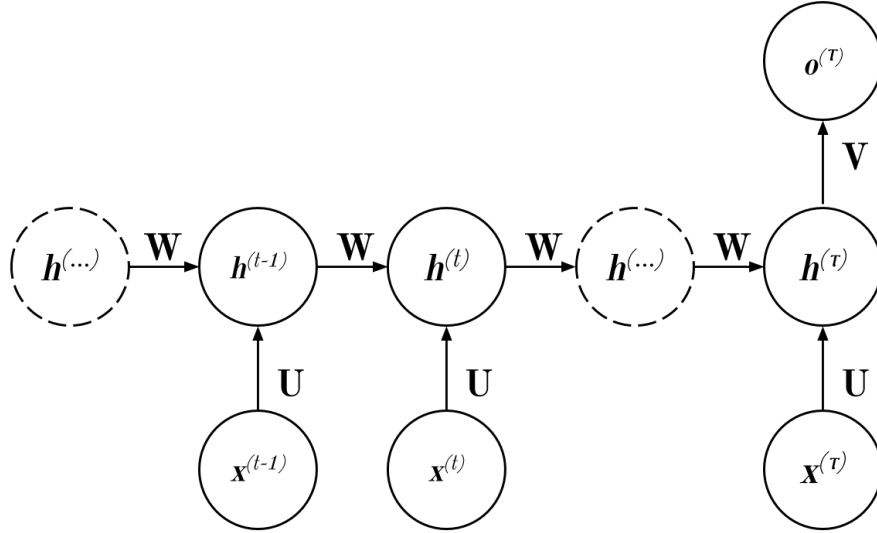


Figura 7: Una red recurrente [32] *unfolded* con una sola salida al final de la secuencia.

El cálculo de los valores de salida \mathbf{o} de la red se realiza utilizando *forward propagation*. Si se observa la Figura 7, se puede ver cómo se calcula utilizando un conjunto de parámetros de la red.

También se puede usar una red que tenga una salida por paso de tiempo (Figura 9). Este tipo de red se conoce como *Elman network* [25]. Otra variación interesante con la que se puede experimentar, es cambiar la recurrencia y, en vez de realimentar el estado con el anterior, se puede utilizar la salida anterior (Figura 10). Este tipo de red se conoce como *Jordan network* [49].

Las ecuaciones que se usan para el cálculo de la salidas para cada paso del tiempo, para el caso de la *Elman Network* [25] (Figura 9), son las siguientes:

$$\mathbf{h}^{(t)} = \theta(\mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{h}^{(t-1)} + \mathbf{b}^h) \quad (\text{Ec. 2.10})$$

$$\mathbf{o}^{(t)} = \mathbf{V}\mathbf{h}^{(t)} + \mathbf{b}^o \quad (\text{Ec. 2.11})$$

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{o}^{(t)}) \quad (\text{Ec. 2.12})$$

Los parámetros que se utilizan en las ecuaciones, los vectores de bias \mathbf{b}^h y \mathbf{b}^o , y las matrices \mathbf{U} , \mathbf{W} y \mathbf{V} , además del estado inicial $\mathbf{h}^{(0)}$, son los que luego serán ajustados con el proceso de *backpropagation* de la red. La Figura 8 muestra la representación gráfica de una celda en una red RNN.

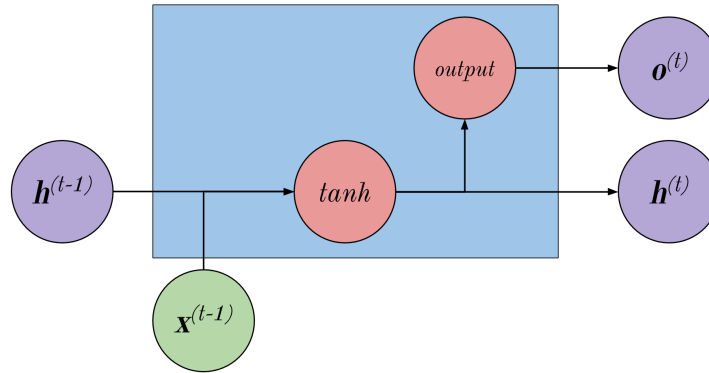


Figura 8: Una celda de una red RNN, donde se puede ver el procesamiento de un paso de la secuencia.

Se puede calcular una salida por paso, y en el caso de la red con una sola salida (Figura 7) solamente es necesario calcular ésta para el último paso de la red, donde $t = \tau$. Para el caso de una *Jordan Network* (Figura 10), la única ecuación a modificar es la que calcula el estado oculto $\mathbf{h}^{(t)}$ (Ec. 2.10). El concepto detrás de este cambio es que, si cada salida es una acción, esto permite a la red recordar las acciones realizadas y que éstas sean tomadas en cuenta a la hora de tomar la próxima acción. Esto puede ser aplicado en varias tareas, como por ejemplo a la hora de traducir una oración entre lenguajes naturales [88], donde es importante tener como entrada la salida anterior, que en ese caso será una secuencia de texto. La ecuación se modifica utilizando como entrada en vez del estado oculto anterior $\mathbf{h}^{(t-1)}$, la salida anterior $\mathbf{o}^{(t-1)}$:

$$\mathbf{h}^{(t)} = \theta(\mathbf{U}\mathbf{x}^{(t)} + \mathbf{W}\mathbf{o}^{(t-1)} + \mathbf{b}^h) \quad (\text{Ec. 2.13})$$

Cada uno de los parámetros se encuentra asociado a una de tres transformaciones de una red RNN, de la entrada a un estado oculto (\mathbf{U}), de un estado oculto al próximo estado oculto (\mathbf{W}) y de un estado oculto a la salida (\mathbf{V}). Cada uno de ellos puede estar asociado a una transformación que podemos llamar *shallow*, que es una transformación posible en una capa de una red MLP (Ec. 2.1).

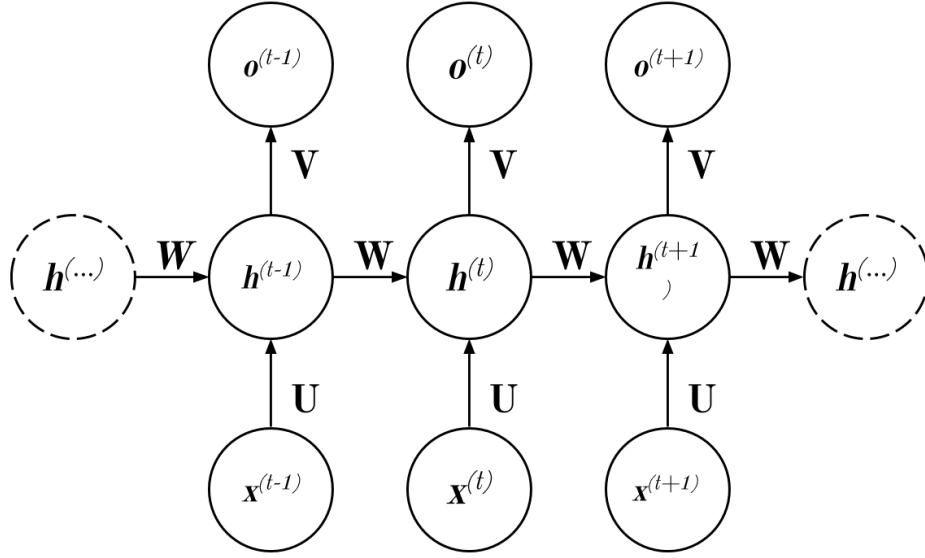


Figura 9: Una red recurrente [32] *unfolded* conocida como *Elman network* [25].

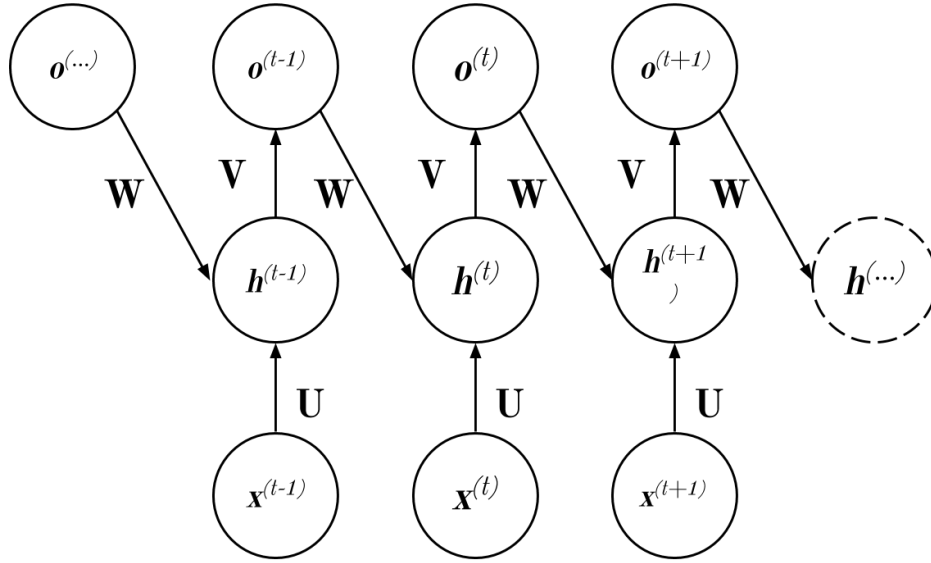


Figura 10: Una red recurrente [32] *unfolded* conocida como *Jordan network* [49].

Existen dos problemas ampliamente conocidos a la hora de entrenar redes RNN, los cuales son denominados : *exploding gradient problem* [3, 42, 43] y *vanishing gradient problem*. El *exploding gradient problem* se refiere a un gran incremento en la norma del gradiente durante el entrenamiento, causado por la explosión de los componentes de largo plazo. Esto puede resultar en inestabilidad en el proceso de aprendizaje. Análogamente, *vanishing gradient problem* se refiere al comportamiento opuesto, donde el componente de largo plazo converge exponencialmente a una norma cero, volviendo imposible que la red aprenda correlaciones, entre eventos distantes temporalmente, eliminando una

de las ventajas de este tipo de redes. Esto refleja de alguna manera una contrapartida entre el aprendizaje por *gradient descent* y la capacidad de poder capturar dependencias a medida que la duración de las mismas aumentan a través del tiempo. Este problema no se presenta únicamente en las redes RNN pero, debido a la profundidad de este tipo de redes y la forma en la cual funciona su algoritmo de feedforward, son unas de las mas afectadas por este fenómeno.

Una de las variantes para aliviar este problema es la de usar una función de activación ReLU (Ec. 2.2)[31, 96], debido a la forma que toma la salida, el gradiente de la función ReLU sólo puede tomar como valores 0 o 1, evitando la saturación y por lo tanto el efecto de desvanecimiento o explosión. Otras maneras de solucionar este problema, además de modificar la función de activación, es utilizar otros tipos de redes recurrentes como la red LSTM [44] (ver Apéndice, sección A.2) o la red GRU [13] (ver Apéndice, sección A.3) o hacer *gradient clipping* [77], limitando el crecimiento del gradiente. Una alternativa, que también es muy utilizada, es modificar la inicialización de los parámetros. Ésta es un área muy investigada, ya que, en la práctica, se notó que afecta al rendimiento de la red de manera considerable. Se demostró que inicializar los valores de la red como una matriz identidad es una técnica efectiva para combatir el problema de los gradientes que desaparecen, y las redes así inicializadas se las llama IRNN (identity RNN) [62]. Otros métodos conocidos de inicialización son la *xavier initialization* [30] o la *kaiming initialization* [38].

CAPÍTULO 3: DESCOMPOSICIONES TENSORIALES

Este capítulo se enfoca en las descomposiciones tensoriales, con una introducción teórica a los tensores, donde se especifican tanto la notación como las operaciones que se utilizan a lo largo del trabajo. Luego se hace un repaso histórico de las descomposiciones tensoriales y sus aplicaciones actuales en diferentes campos. También se explican en detalle dos descomposiciones que, más adelante en el trabajo, serán aplicadas en redes neuronales profundas: Tensor Ring (TR) y Tensor Train (TT).

Un tensor es la generalización multidimensional de un vector o matriz [57]. Este término generalmente se utiliza para referirse a un arreglo que tiene tres o más índices (tensores de orden alto), pero también puede usarse para describir una matriz (orden 2), un vector (orden 1), o un escalar (orden 0). El uso de tensores es común en varias áreas, pero no debe ser confundido por los usados en campos como la física y la ingeniería, comúnmente llamados *campos tensoriales*. Un subtensor es una parte del tensor original que se obtiene fijando un subconjunto de índices.

Los subtensores de orden 1, que son vectores llamados *fibers*, se obtienen fijando todos menos un índice, mientras que los tensores de orden 2 son matrices llamados *slices*, y se obtienen fijando todos los índices menos dos. Los tensores denominados simples son aquellos tensores que pueden ser escritos como un producto de tensores de orden 1:

$$\mathcal{X} = \mathbf{b}_r^{(1)} \otimes \mathbf{b}_r^{(2)} \otimes \dots \otimes \mathbf{b}_r^{(N)} \quad (\text{Ec. 3.1})$$

Los tensores tienen, asimismo, un rango que se define como la cantidad mínima necesaria de tensores simples (también denominados de rango 1) que sumados dan como resultado dicho tensor.

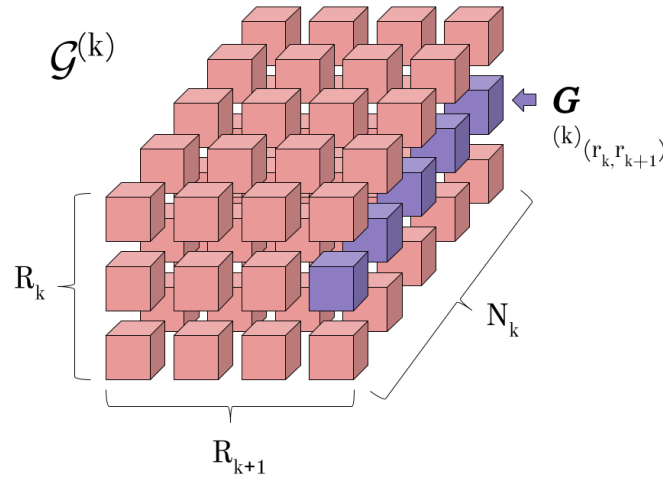


Figura 11: Ejemplo de un tensor $\mathcal{G}^{(k)} \in \mathbb{R}^{R_k \times N_k \times R_{k+1}}$ de orden 3, con una *fiber* $\mathbf{g}^{(k)}(r_k, r_{k+1})$.

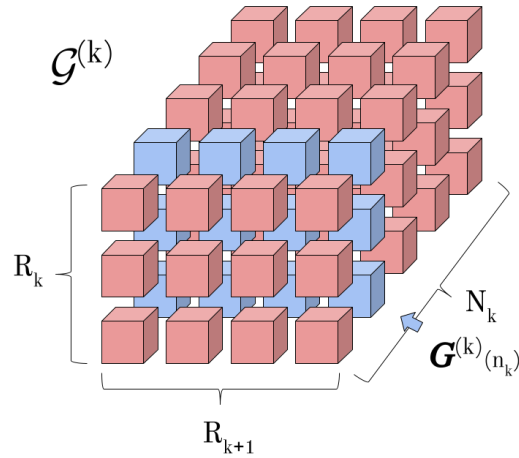


Figura 12: Ejemplo de un tensor $\mathcal{G}^{(k)} \in \mathbb{R}^{R_k \times N_k \times R_{k+1}}$ de orden 3, con una *slice* $\mathbf{G}^{(k)}(n_k)$.

3.1. Notación de Tensores

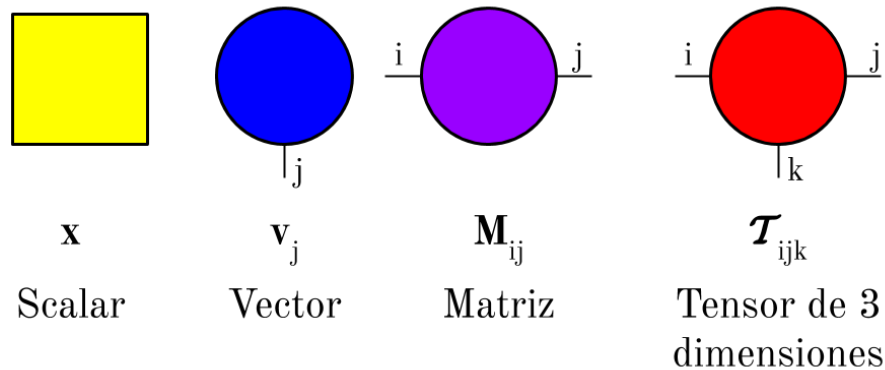
Notación básica	
$\mathcal{A}, \mathbf{A}, \mathbf{a}, a$	Tensor, Matriz, Vector, Escalar
$\mathcal{D} = \text{diag}_N(\lambda_1, \lambda_2 \dots, \lambda_N)$	Tensor diagonal de orden N
$\mathbf{D} = \text{diag}_N(\lambda_1, \lambda_2 \dots, \lambda_N)$	Matriz diagonal de orden N
$\mathbf{B}^{(i)}$	Matriz factor i
$\mathbf{b}_j^{(i)}$	Vector columna j de la matriz $\mathbf{B}^{(i)}$
$a_{r_1 r_2 \dots r_N}$	Elemento de un tensor $\mathcal{A} \in \mathbb{R}^{R_1 \times R_2 \times \dots \times R_N}$
$\mathcal{G}^{(i)}$	Tensor núcleo número i de orden 3 con índices j, k, l para cada uno de sus dimensiones
$\mathbf{G}^{(i)}(j)$	<i>Slice</i> de un Tensor Núcleo $\mathcal{G}^{(i)}$ de orden 3 con el índice j fijado
$\mathbf{g}^{(i)}(j, k)$	<i>Fiber</i> de un Tensor Núcleo $\mathcal{G}^{(i)}$ de orden 3 con los índices j y k fijados
$\mathbf{A}^T, \mathbf{a}^T$	Matriz transpuesta, Vector transpuesto

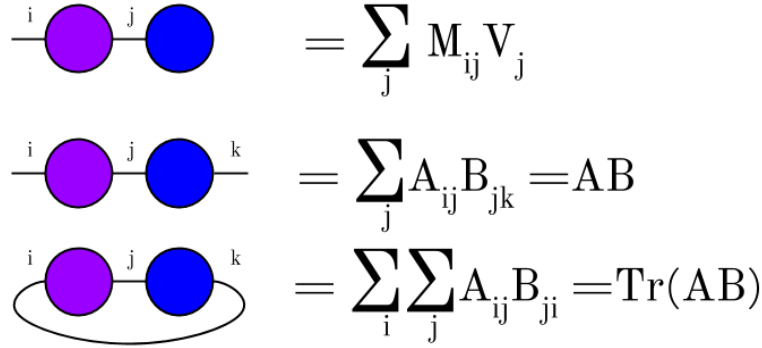
Tabla 1: Notación básica de Tensores

3.1.1. Notación de Diagrama de Tensores

Hay varios sistemas de notación gráfica utilizados para tensores y redes de los mismos. En este documento se utilizarán las siguientes convenciones [90]:

- Un tensor es notado como un círculo sólido y sus índices son las líneas que emanan de éste (Figura 13).
- Una línea que conecta dos implica una contracción o suma sobre los índices conectados (Figura 14).





$$\begin{aligned}
 & \text{Diagram 1: } \text{Purple circle (i)} - \text{Blue circle (j)} = \sum_j M_{ij} V_j \\
 & \text{Diagram 2: } \text{Purple circle (i)} - \text{Blue circle (j)} - \text{Blue circle (k)} = \sum_j A_{ij} B_{jk} = AB \\
 & \text{Diagram 3: } \text{Purple circle (i)} - \text{Blue circle (j)} - \text{Blue circle (k)} \text{ with a loop} = \sum_i \sum_j A_{ij} B_{ji} = \text{Tr}(AB)
 \end{aligned}$$

Figura 14: Ejemplos de notación de diferentes contracciones. El primer ejemplo es el producto de una matriz por un vector, el siguiente es el de una matriz por otra matriz y el tercero es la traza del producto.

3.2. Historia

Fue Hitchcock en 1927 [40, 41] quien primero introdujo la idea de descomponer tensores, demostrando que se pueden representar a través de un conjunto de tensores o vectores de menor orden. Hitchcock propuso descomponer un tensor como la suma de un producto de vectores. Esto más los aportes de Catell en 1952 [10], fueron la inspiración para que en los años 70s se desarrollaran las descomposiciones CANDECOMP (Descomposición canónica) [9] y PARAFAC [36]. Éstas son la base de una de las descomposiciones más conocidas y estudiadas, la CANDECOMP/PARAFAC [53], también llamada la descomposición poliádica canónica (CPD) o tensor de Kruskal (Figura 15). La CPD es utilizada para hacer procesamiento y análisis de señales, o en análisis exploratorio de datos. La CPD representa un tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ de orden N como una combinación lineal de tensores de rango 1:

$$\mathcal{X} = \sum_{r=1}^R \lambda_r \cdot (\mathbf{b}_r^{(1)} \otimes \mathbf{b}_r^{(2)} \otimes \dots \otimes \mathbf{b}_r^{(N)}) \quad (\text{Ec. 3.2})$$

Esta ecuación también se puede escribir como un producto multilinear con un núcleo diagonal $\mathcal{D} = \text{diag}_N(\lambda_1, \lambda_2, \dots, \lambda_R)$:

$$\begin{aligned}
 \mathcal{X} &= \mathcal{D} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \dots \times_N \mathbf{B}^{(N)} \\
 &= \|\mathcal{D}; \mathbf{B}^{(1)}, \mathbf{B}^{(2)}, \dots, \mathbf{B}^{(N)}\|
 \end{aligned} \quad (\text{Ec. 3.3})$$

Kruskal en 1977 [61] propuso el concepto de rango de un tensor, análogo al rango de una matriz, que se define como la menor cantidad de tensores de rango 1 (tensores que son producto de vectores) que sumados dan ese mismo tensor. Notablemente, hallar el rango de un tensor, también llamado rango canónico, es NP-Hard [37], inclusive para un tensor de orden 3. Esto, junto a la dificultad de encontrar una buena aproximación de tensores de bajo rango, inclusive cuando es sabido que éste existe, suele disuadir el uso de este formato. Por dicha razón también se incentivó el uso de otros modelos de descomposición, a pesar de tener más parámetros que la canónica, ya que son más fáciles de lidiar numéricamente. Aún así es importante notar la baja cantidad de parámetros del modelo CPD $\mathcal{O}(dnr)$, siendo d el orden del tensor, n el tamaño de cada uno de los vectores y r el rango.

Las descomposiciones tensoriales volvieron a popularizarse debido a su utilidad a la hora de realizar análisis de datos, sobre todo con grandes volúmenes de datos (big data), que son representados de manera multidimensional. Es una más de las herramientas que se utilizan a la hora de tratar de

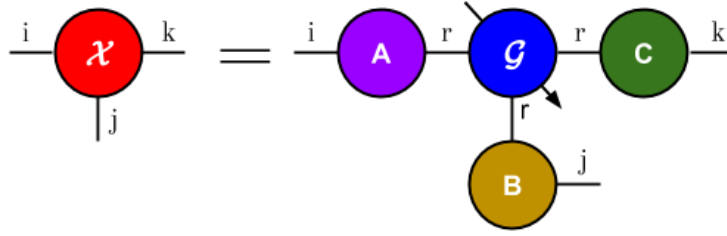


Figura 15: Ejemplo de una descomposición CPD de un tensor \mathcal{X} de orden 3 de dimensiones $I \times J \times K$, en el producto de 3 matrices, \mathbf{A} ($I \times R$), \mathbf{B} ($I \times R$) y \mathbf{C} ($K \times R$) por el tensor diagonal \mathcal{G} también de orden 3 ($R \times R \times R$).

resolver la maldición de la dimensionalidad (*curse of dimensionality*) [2]. Los datos a analizar pueden venir naturalmente en forma de un tensor, como por ejemplo, una imagen o un vídeo, o pueden ser datos representados en formatos más simples, como un vector o una matriz, tensorizados con el fin de poder organizarlos de manera más simple o de poder darle más sentido a su representación, como por ejemplo, las mediciones de un grupo de sensores a través del tiempo o respecto de sus canales.

Las descomposiciones proveen la capacidad de realizar separación ciega de fuentes (BSS), es decir, separar señales de diferentes fuentes provenientes de varios sensores. Éstas también son capaces de realizar reducción de dimensionalidad y lidiar datos con valores faltantes o ruidosos. Son una buena herramienta para analizar la relación e interacción entre diferentes dimensiones o modos de los datos, pudiendo las redes tensoriales tomar múltiples formas que pueden adaptarse dependiendo de su naturaleza. Otra aplicación que será mencionada más adelante es su poderosa capacidad de compresión, pasando de datos multidimensionales a una factorización en matrices o tensores de muy bajo rango y orden. Esto también permite que sea más fácil y menos costoso realizar ciertas operaciones matemáticas, que al mismo tiempo, debido a la forma de las redes, se vuelven más fácil de interpretar.

La descomposición de Tucker, introducida en 1963 [92], consiste en un tensor núcleo pequeño que es multiplicado por matrices (factores) en cada dimensión (Figura 16). Suele ser usada para el análisis de componentes principales (PCA) de orden alto. El problema de éste tipo de descomposición es que la cantidad de parámetros que requiere son $\mathcal{O}(dnr + r^d)$, teniendo un término exponencial que, para un tensor de orden grande, se vuelve prohibitivo. Esta descomposición funciona para dimensiones chicas, especialmente para el caso de un tensor con 3 dimensiones [76]. Además de su uso como PCA, tiene otros usos interesantes como en *computer vision* [93], o en el reconocimiento de dígitos manuscritos [83].

La descomposición de Tucker representa un tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ de orden N como una transformación multilinear de un tensor núcleo $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \dots \times R_N}$ más denso y pequeño del original multiplicado por matrices $\mathbf{B}^{(n)} = [\mathbf{b}^{(1)}, \mathbf{b}^{(2)}, \dots, \mathbf{b}^{(n)}] \in \mathbb{R}^{I_n \times R_n}$, $n = 1, 2, \dots, N$, dado por:

$$\mathcal{X} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_N=1}^{R_N} g_{r_1 r_2 \dots r_N} \cdot \left(\mathbf{b}_{r_1}^{(1)} \otimes \mathbf{b}_{r_2}^{(2)} \otimes \dots \otimes \mathbf{b}_{r_N}^{(N)} \right), \quad (\text{Ec. 3.4})$$

o equivalentemente:

$$\begin{aligned} \mathcal{X} &= \mathcal{G} \times_1 \mathbf{B}^{(1)} \times_2 \mathbf{B}^{(2)} \dots \times_N \mathbf{B}^{(N)} \\ &= \left\| \mathcal{G}; \mathbf{B}^{(1)}, \mathbf{B}^{(2)}, \dots, \mathbf{B}^{(N)} \right\| \end{aligned} \quad (\text{Ec. 3.5})$$

En la representación de la descomposición de Tucker como un producto multilinear (Ec. 3.5) comparándola con la representación de la CPD (Ec. 3.3), podemos ver que la última es un caso especial de

la anterior, es decir con un tensor núcleo \mathcal{G} diagonal y con todas sus dimensiones iguales.

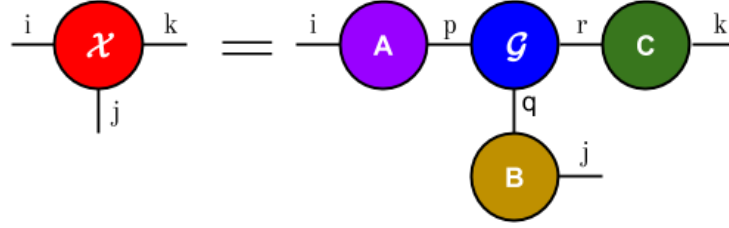


Figura 16: Ejemplo de una descomposición de Tucker de un tensor \mathcal{X} de orden 3 de dimensiones $I \times J \times K$, en el producto de 3 matrices, \mathbf{A} ($I \times P$), \mathbf{B} ($I \times Q$) y \mathbf{C} ($K \times R$) por el tensor \mathcal{G} también de orden 3 ($P \times Q \times R$). Es importante notar que a diferencia de CPD, donde el núcleo es diagonal, en Tucker es denso.

De la descomposición de Tucker surge otra llamada Hierarchical Tucker (HT), desarrollada por Hackbusch *et al.* [35], que realiza el mismo proceso, pero de manera jerárquica, lo que lo vuelve más complejo y compacto. Este tipo de descomposiciones, donde el número de tensores y matrices involucrados en la descomposición es grande, se conoce como Redes Tensoriales (*Tensor Networks*) y pueden representarse sintéticamente a través de diagramas de redes. Por ejemplo, la Figura 17 muestra la red asociada a una descomposición HT.

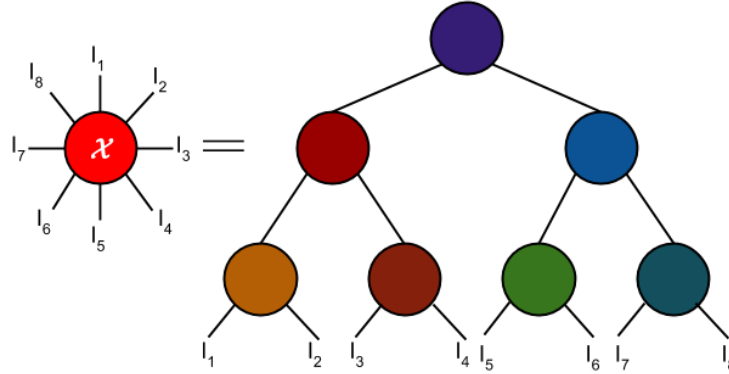


Figura 17: Ejemplo de una descomposición HT de un tensor \mathcal{X} de orden 8.

3.3. Tensor Train

Tensor Train (TT) [75] es otra red tensorial que puede ser interpretada como un HT, en el cual todos los nodos están conectados en cascada, haciéndolo un tren de núcleos (Figura 18), de ahí su nombre. Tanto en el caso de TT como el de HT tienen como ventaja que la representación de un tensor de orden alto está reducida a un conjunto d de tensores de orden 2 y 3, y por lo tanto están libres de la maldición de la dimensionalidad. El costo de almacenamiento de la descomposición de TT es de $\mathcal{O}(dnr^2)$. Los dimensiones de entrada y salida de los tensores núcleos son llamados *TT-Ranks* o *rangos* de la descomposición. Éstos solamente tienen como valores restringidos el rango de entrada del primer tensor y el último rango de salida del último tensor, $r_1 = r_{d+1} = 1$, haciendo que esos tensores sean matrices, $\mathcal{G}^{(1)} \in \mathbb{R}^{1 \times I_1 \times R_2}$ y $\mathcal{G}^{(N)} \in \mathbb{R}^{R_N \times I_N \times 1}$. Los otros rangos no están necesariamente restringidos, pero generalmente es común que los rangos de los tensores intermedios sean más grandes que aquellos de los extremos.

La descomposición TT está basada en aproximar cada elemento del tensor por un producto secuencial de matrices, donde la primera y la última matriz son vectores, para asegurarse que el resultado sea escalar. Esta descomposición es muy utilizada en la física, es conocida como MPS (*matrix product state*). Siendo un elemento x_{i_1, i_2, \dots, i_N} del tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, es el producto secuencial de unas matrices de la forma $\mathbf{G}^{(n)}(i_n) \in \mathbb{R}^{R_{n-1} \times R_n}$, una *slice* de un núcleo $\mathcal{G}^{(n)} \forall n = 1 \dots N$:

$$x_{i_1, i_2, \dots, i_N} = \mathbf{G}^{(1)}(i_1) \mathbf{G}^{(2)}(i_2) \dots \mathbf{G}^{(N)}(i_N) \quad (\text{Ec. 3.6})$$

Teniendo en cuenta que $\mathbf{G}^{(1)}(i_1) \in \mathbb{R}^{1 \times R_1}$ y $\mathbf{G}^{(N)}(i_N) \in \mathbb{R}^{R_N \times 1}$ y $g_{r_n, i_n, r_{n+1}}^{(n)}$ un elemento del tensor $\mathcal{G}^{(n)}$ puede escribirse como:

$$x_{i_1, i_2, \dots, i_N} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_N=1}^{R_N} g_{1, i_1, r_2}^{(1)} \cdot g_{r_1, i_2, r_3}^{(2)} \dots g_{r_N, i_N, 1}^{(N)} \quad (\text{Ec. 3.7})$$

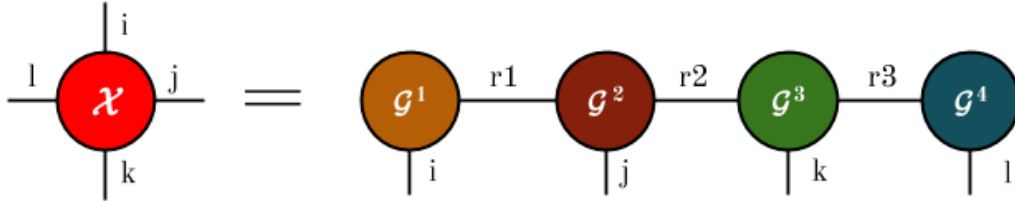


Figura 18: Ejemplo de una descomposición *Tensor Train* de un tensor \mathcal{X} de orden 4.

3.4. Tensor Ring

En la actualidad se siguen desarrollando nuevos tipos de descomposiciones como *Tensor Ring* (TR) [98] (Figura 19), que fue pensada con el propósito de reducir el tamaño de almacenamiento necesario para guardar tensores de alta dimensionalidad. Está basada en TT, con un cambio en la conexión de los núcleos, el primero está conectado con el último, formando justamente un anillo de tensores núcleos. Esto ataca una de las restricciones más importantes de la descomposición TT, los valores de los rangos fijos, $r_1 = r_{d+1} = 1$, pudiendo llevar a limitar la capacidad y flexibilidad de la descomposición. Para atacar ésta y otras restricciones como los patrones de los rangos y el orden específico en el cual se debe realizar el producto multilinear, surgió esta nueva descomposición.

De la misma manera que en la descomposición TT, en este caso los valores de las dimensiones exteriores del tensor $R_n \forall n = 1 \dots N$ son llamados *TR-Ranks* o *rangos* de la descomposición. El principal cambio propuesto es sacar las restricciones de los rangos de tamaño fijo en el primer y último tensor. Debido a que estos rangos pasan a tener valores distintos de uno, se utiliza la traza para reducir la matriz generada al valor de un elemento del tensor. Finalmente, para expresar un elemento del tensor t_{i_1, i_2, \dots, i_N} de un tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, se utiliza la traza del producto secuencial de unas matrices de la forma $\mathbf{G}^{(n)}(i_n) \in \mathbb{R}^{R_{n-1} \times R_n}$ una *slice* de un núcleo $\mathcal{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n} \forall n = 1 \dots N$:

$$\begin{aligned}
 t_{i_1, i_2, \dots, i_N} &= \text{Tr} \left\{ \mathbf{G}^{(1)}(i_1) \mathbf{G}^{(2)}(i_2) \dots \mathbf{G}^{(N)}(i_N) \right\} \\
 &= \text{Tr} \left\{ \prod_{k=1}^N \mathbf{G}^{(k)}(i_k) \right\}
 \end{aligned} \tag{Ec. 3.8}$$

Es importante que cada par de tensores núcleos compartan una dimensión, siendo $\mathcal{G}^{(n)} \in \mathbb{R}^{R_{n-1} \times I_n \times R_n}$ y $\mathcal{G}^{(n+1)} \in \mathbb{R}^{R_n \times I_n \times R_{n+1}} \forall n = 1 \dots N$, lo cual genera que para el tensor N , $R_1 = R_{N+1}$. Esto asegura que el producto del producto secuencial sea una matriz cuadrada, por lo tanto pueda aplicarse la traza de la misma. Una propiedad importante que tiene esta descomposición a diferencia de la TT es que se pueden permutar los tensores núcleos sin cambiar el resultado final. Haciendo un desplazamiento circular de las dimensiones de un tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ de una descomposición dada por $\mathcal{T} = \mathfrak{R}(\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, \dots, \mathcal{G}^{(N)})$ por k entonces se tiene una descomposición \mathcal{T}^k que es igual a la original. Tomando en cuenta (Ec. 3.8) se puede escribir:

$$\begin{aligned}
 t_{i_1, i_2, \dots, i_N} &= \text{Tr} \left\{ \mathbf{G}^{(2)}(i_2) \mathbf{G}^{(3)}(i_3) \dots \mathbf{G}^{(N)}(i_N), \mathbf{G}^{(1)}(i_1) \right\} \\
 &= \dots = \text{Tr} \left\{ \mathbf{G}^{(N)}(i_N) \mathbf{G}^{(1)}(i_1) \dots \mathbf{G}^{(N-1)}(i_{N-1}) \mathbf{G}^{(1)}(i_1) \right\} \\
 &\Rightarrow \mathcal{T}^k = \mathfrak{R}(\mathcal{G}^{(k+1)}, \dots, \mathcal{G}^{(N)}, \mathcal{G}^{(1)}, \dots, \mathcal{G}^{(k)})
 \end{aligned} \tag{Ec. 3.9}$$

Esta propiedad (Ec. 3.9) tiene como consecuencia que, a diferencia de la descomposición TT, no hace falta elegir una permutación dimensional óptima, permitiendo también que sea más flexible, ya que no requiere un patrón fijo de rangos. Si uno toma como referencia la ecuación (Ec. 3.7), se puede llegar a la conclusión que la descomposición TT es en realidad un caso especial de la TR. Se puede pensar que, en ese caso, la matriz sobre la que se aplica la traza termina siendo un escalar. Se define los *TR-Ranks* como $R_1 = R_{N+1} = 1$ se puede pasar simplemente desde la ecuación (Ec. 3.8) a la (Ec. 3.7):

$$\begin{aligned}
 t_{i_1, i_2, \dots, i_N} &= \text{Tr} \left\{ \mathbf{G}^{(1)}(i_1) \mathbf{G}^{(2)}(i_2) \dots \mathbf{G}^{(N)}(i_N) \right\} \\
 &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_N=1}^{R_N} z_{1, i_1, r_2}^{(1)} \cdot z_{r_2, i_2, r_3}^{(2)} \dots z_{r_N, i_N, 1}^{(N)}
 \end{aligned} \tag{Ec. 3.10}$$

Esta última ecuación, además de demostrar que la descomposición TR es una generalización de la TT, también permite la interpretación de que un modelo TR puede ser representado como una combinación lineal de varias TT, cuyos núcleos son parcialmente compartidos. Esto explica porque los *TR-Ranks* sean menores que los *TT-Ranks*. También se puede demostrar que las descomposiciones CPD y de Tucker son casos específicos de una descomposición TR, para ver la demostración referirse a Zhao *et al.* [98].

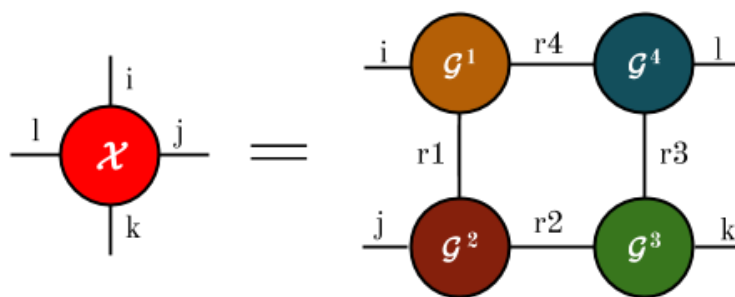


Figura 19: Ejemplo de una descomposición *Tensor Ring* de un tensor \mathcal{X} de orden 4.

CAPÍTULO 4: EL ROL DE LAS DESCOMPOSICIONES TENSORIALES EN DEEP LEARNING

En este capítulo se profundiza en detalle cómo se utilizan las descomposiciones tensoriales en el área de *deep learning*. Abarcando un repaso histórico de los primeros trabajos en este campo hasta el estado del arte, empezando por su utilización como herramienta en el proceso de compresión de capas de redes neuronales profundas de diferente tipo. Luego se describe el concepto de expresividad de una red neuronal y cómo es la relación entre diferentes arquitecturas y descomposiciones. Finalmente, se detalla la arquitectura propuesta y como ésta se compara con arquitecturas utilizadas en la literatura reciente.

4.1. Compresibilidad de redes neuronales profundas

Debido a la popularidad de arquitecturas de *deep learning* para resolver diferentes tipos de problemas, principalmente de redes neuronales convolucionales (CNN) [65], surgió el interés de poder desplegar este tipo de modelos en dispositivos de menor potencia computacional. En un principio, estos modelos se corrían en equipamiento con mucha capacidad, como clusters de *Graphics Processing Units* (GPU), de *Central Processing Units* (CPU) o Field-Programmable Gate Arrays (FPGA). A la hora de correr estas redes, tanto para entrenarlas como para ejecutarlas en laptops o celulares, pueden resultar muy demandantes en cuanto a tiempo y cálculo, haciendo su uso prohibitivo. El problema también se refleja a la hora de hacer desarrollos basados en la web, donde modelos en la nube deben procesar millones de entradas por día. Esto puede conllevar grandes costes computacionales. Una de las múltiples maneras de abordar este problema es enfocándose en las capas convolucionales. Éstas son particularmente atractivas de comprimir porque usualmente requieren de un gran número de parámetros. Esta sobreparametrización, a su vez, facilita la convergencia a un buen mínimo local de la función de costo durante el entrenamiento [21].

El trabajo de Denton *et al.* 2014 [22] propone explotar esta redundancia en las capas convolucionales, utilizando técnicas de compresión lineal sobre las mismas. En el trabajo se aplican descomposiciones tensoriales para conseguir una aproximación lineal de bajo rango sobre las capas convolucionales de una red ya entrenada. Luego de ser comprimida, se realiza un ajuste sobre los parámetros de las capas afectadas para restaurar el correcto funcionamiento de las mismas. Ésto tiene como resultado una reducción importante en la cantidad de parámetros y en el tiempo de procesamiento de la red, sin comprometer el rendimiento de la misma.

En su trabajo, Jaderberg *et al.* 2014 [48] planteó una arquitectura parecida, donde se cambian capas convolucionales con una combinación lineal de capas simples, argumentando que se pueden explotar redundancias que existen entre diferentes canales de *features* y filtros. También vale la pena mencionar el trabajo de Rigamonti *et al.* 2013 [80], el cual se basa en una idea parecida, la de poder aprender las redundancias de los filtros. El trabajo de Jaderberg *et al.* [48] fue reutilizado en Tai *et al.* 2015 [89], que usa un algoritmo diferente de descomposición para realizar la misma conversión de una capa convolucional en un conjunto de capas más simples.

Siguiendo en la temática de compresión de redes ya pre entrenadas [63], un trabajo de 2015 [74] utilizó la misma metodología que en trabajos anteriores, tensorizando las capas de una CNN [85]. En este caso [74] se utiliza una descomposición TT [75] sobre las capas completamente conectadas y se denomina a esa capa transformada como una *TT-Layer*, que, junto al resto de las capas, forma una red denominada *TensorNet*. Las capas completamente conectadas de manera similar a las capas convolucionales se caracterizan por sus múltiples redundancias. Las mismas pueden ser eliminadas por una descomposición TT, que tiene las ventajas de ser un algoritmo más avanzado y de ser inmune a la maldición de la dimensionalidad a diferencia de la CPD. Las capas llamadas *TT-Layers* son compatibles con los algoritmos de entrenamiento existentes, ya que las derivadas requeridas por el algoritmo de *backpropagation* pueden ser calculadas usando las propiedades de la descomposición TT. Las redes que utilizan las *TT-layers* empatan la performance de las redes sin este tipo de capas, pero requieren menos de 7 veces la cantidad de parámetros de la red completa [74].

El concepto de Novikov *et al.* 2015 [74] fue profundizado en otro paper por el mismo equipo en 2016 [28], ésta vez abordando las capas convolucionales, volviendo a utilizar la descomposición TT [75] para comprimirlas. En el trabajo se propone reutilizar el enfoque anteriormente empleado, comprimir las capas completamente conectadas tratándolas como un tensor de orden alto y utilizando la descomposición para poder representarla como una serie de tensores de orden bajo, en este caso además se propone descomponer las capas convolucionales. Los resultados de éstos dos papers combinados logran reducir de manera importante la cantidad de parámetros de CNN (80x) sin

reducir significativamente la exactitud (1.1 %) en una red que tiene ambos tipos de capas reemplazadas por TT-Layers.

En el trabajo de Kim *et al.* [54] en 2015 se introduce un enfoque diferente para comprimir CNN, mediante un esquema que llaman “compresión completa de la red en una sola oportunidad”. Este enfoque varía respecto a los previamente mencionados, en que en los anteriores trabajos se enfocan en ir descomponiendo y reajustando las capas de manera individual, en este caso toma la idea de hacer todo el proceso de manera paralela, haciendo la descomposición y la sintonía fina en cada una de las capas al mismo tiempo. Esto se realiza en 3 pasos secuenciales: (1) la utilización de un método para elegir los rangos de la descomposición de cada una de las capas, (2) la aplicación de la descomposición de Tucker en cada una con el rango previamente determinado y (3) la realización de la sintonía fina sobre todos los coeficientes de la red, utilizando el algoritmo de backpropagation, con una serie de CNN profundas pre entrenadas cuyos resultados en smartphones fueron buenos.

Uno de los trabajos más novedosos, es el realizado por Kossaifi *et al.* [58], que, continuando con la línea de trabajos, utiliza descomposiciones para poder utilizar tensores de bajo orden para que aproximen a los de orden alto que representan las capas, tanto convolucionales como las completamente conectadas de una red CNN. En este caso se introducen dos tipos de capas diferentes para poder realizar este proceso, la primera, llamada capa de contracción tensorial (TLC), que aplica una contracción a un tensor de activación para poder obtener una representación de baja dimensionalidad, y la segunda, denominada capa de regresión tensorial (TRL), que expresan salidas a través de un mapeo multilineal de rango bajo desde un tensor de orden superior a un tensor de salida de orden arbitrario. Reemplazando la última operación de *pooling*, aplanamiento y las capas completamente conectadas con una combinación de TLC y TRL en CNN, se obtiene como resultado redes de menos parámetros con un nivel parecido de performance.

Las descomposiciones también han sido utilizadas en el campo de modelos generativos, siendo implementadas en redes generativas adversarias (GAN) [33], que demuestran un muy buen rendimiento en este campo. Las redes GAN requieren de modelos muy grandes para obtener buenos resultados. La necesidad de modelos tan complejos está en parte atribuida a la naturaleza multidimensional de los bases de datos con los cuales trabaja. El caso de las GAN es similar ya que se topa con el problema de utilizar MLP para aprender y clasificar estas bases, lo cual implica el surgimiento de varios de los problemas que ya anteriormente mencionados. La inmensa cantidad de parámetros requeridos y las operaciones vectorizadas generan pérdida de información debido a la pérdida de dimensionalidad en los datos. La red generativa consiste básicamente de 2 componentes llamados generador y discriminador. La labor del discriminador es la de distinguir si su entrada proviene de un base de datos original o de los fabricados por el generador. El objetivo del generador es el de engañar al discriminador, tratando de generar muestras indistinguibles del set original. Basándose en los trabajos anteriormente mencionados [58, 74] Cao *et al.* [8] utiliza capas que llaman *tensor layers*, para reemplazar la red MLP que suelen ser la implementación tanto del discriminador como del generador de la red GAN. Las *tensor layers* son capas donde se reemplazan los tensores de pesos por una descomposición de Tucker, utilizando este tipo de capas los autores concluyen que se puede reducir la cantidad de recursos mientras que se gana la capacidad de capturar datos con estructuras multidimensionales.

Otro trabajo que vincula la descomposición TT con redes RNN es Yang *et al.* 2017 [97], que las utiliza para hacer clasificación de vídeo. A diferencia de los enfoques más clásicos, en vez de preprocesar las imágenes con una CNN se expone directamente a la red recurrente a los píxeles de las imágenes, intentando que la red pueda capturar la correlación entre los patrones temporales y los patrones espaciales. Para poder atacar la dificultad de tener una matriz de pesos muy grande para que haga el mapeo para las capas internas de la red se utiliza una descomposición TT.

4.2. Expresividad

En el principio del trabajo se define como objetivo principal de las redes neuronales profundas la aproximación de una función. Una de las propiedades esenciales es que pueden aproximar cualquier función siempre que no se le impongan restricciones en el número de parámetros, a esta propiedad se la llama *universalidad*. A partir de esto se define como eficiencia de profundidad al fenómeno por el cual una función aproximada por una red neuronal profunda de tamaño polinomial requiere una red neuronal superficial (de una sola capa) de tamaño exponencial para obtener la misma precisión. Esta teoría es mencionada en varios trabajos como el poder expresivo de la profundidad o la expresividad de las redes profundas.

Basados en los avances de las descomposiciones, Cohen *et al.* 2016 [18] utilizó HT como herramienta a la hora de dar un marco teórico a esta conjetura de que las redes profundas son exponencialmente más expresivas que las redes superficiales. La hipótesis se basa en evidencia empírica, que dado una cantidad limitada de recursos, en este caso podrían ser neuronas, mientras más profunda es la red, el rendimiento es mejor. La diferencia con los trabajos realizados hasta el momento es que éste aplica su teoría a una red Convolutiva y la compara con una red *fully connected*. El trabajo introduce lo que los autores denominan *circuitos aritméticos convolucionales*, estas redes que tienen dos tipos de nodos: los de suma, donde se hace una suma ponderada por pesos de las entradas, y los de producto, donde realiza el producto de las entradas. En estos dos tipos de nodos, el de sumas implementa la convolución (localidad y compartir) y el de productos implementa el *pooling*, permitiendo que estos circuitos se vuelvan equivalentes a redes llamadas *SimNets* [19], una generalización de una CNN.

Teniendo en cuenta el problema de clasificar una entrada $\mathbf{X} = (\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)}), \mathbf{x}^{(t)} \in \mathbb{R}^n$, en una de las categorías $\mathbf{y} = (1, \dots, Y)$. La clasificación es llevada a cabo a través de una función *Softmax* utilizando como salida la valoración de una *score function* por categoría h_y para toda $y \in \mathbf{y}$. El desafío que se plantea es el mismo que el de este trabajo, el de reducir el orden del tensor de coeficientes de orden $\mathcal{O}(M^N)$ definido como $\mathcal{W}^{(y)}$ utilizando una descomposición tensorial, en el caso del trabajo referenciado utilizando *circuitos aritméticos convoluciones*. Siendo $f_{\theta_{d_1}} \dots f_{\theta_{d_M}}$ las funciones de representación definidas para cada vector $\mathbf{x}^{(1)}$ de la entrada se puede definir una *score function* como:

$$\ell^y(x_1, \dots, x_N) = \sum_{d_1 \dots d_N=1}^M \mathcal{W}_{d_1 \dots d_N}^{(y)} \prod_{i=1}^N f_{\theta_{d_i}}(x_i) \quad (\text{Ec. 4.1})$$

En el trabajo se demuestra que se puede utilizar una descomposición CPD de rango 1 para poder representar como una red superficial, utilizándola para descomponer los tensores de coeficientes $\mathcal{W}^{(y)}$ como:

$$\mathcal{W}^{(y)} = \sum_{g=1}^G a_z^{(y)} \cdot (a_z^{(1)} \otimes \dots \otimes a_z^{(N)}) \quad (\text{Ec. 4.2})$$

Donde $\mathbf{a}^y \in \mathcal{R}^G$ y $\mathbf{a}_z^{(i)} \in \mathcal{R}^M, i \in N, g \in G$, estos tensores son compartidos a través de todas las clases y , utilizando esta definición para reemplazarlo en la expresión en la *score function* (Ec. 4.1):

$$\ell^y(x_1, \dots, x_N) = \sum_{g=1}^G a_z^{(y)} \prod_{i=1}^N \left(\sum_{d=1}^M a_z^{(i)} f_{\theta}(x^{(i)}) \right) \quad (\text{Ec. 4.3})$$

Esta red denominada *CP model* es un *circuito aritmético convolucional* superficial que realiza una descomposición CPD. En el trabajo se demuestra teóricamente que esta red tiene la propiedad de *universalidad*. El siguiente paso en el trabajo de Cohen es mostrar que una red profunda, correspondiente a una descomposición HT [35], que también es universal, puede representar cualquier conjunto de tensores $\mathcal{W}^{(y)}$ descompuestos por el *CP model* con el *HT model* solamente con una

penalidad polinomial en cuanto a recursos. Luego se demuestra que el *HT model* en casi todos los casos genera tensores que requieren un orden exponencial de recursos en el otro modelo. Puesto de otra manera, si uno fija los pesos de un *HT model* a través de una distribución de probabilidad continua, los tensores resultantes no pueden ser aproximados por un *CP model* de tamaño polinómico.

La conclusión a la que el trabajo llega es que las funciones implementables en una red neuronal profunda (excepto una cantidad despreciable) requieren una red superficial de un tamaño exponencialmente más grande para poder ser representada por las mismas, demostrando teóricamente la definición de poder expresivo de la profundidad delineada previamente.

El mismo equipo en un trabajo subsiguiente, *Cohen & Shashua 2016* [20], continuando con esta línea de investigación, estudió las redes denominadas redes rectificadoras convolucionales y su modelado como descomposiciones tensoriales generalizadas. Las redes rectificadoras convolucionales son CNN con activación lineal rectificada y max o promedio *pooling*. Éstas son más utilizadas actualmente. El objetivo de este trabajo es análogo al previamente mencionado, lograr comprender las propiedades de expresividad de este tipo de redes, de la misma manera que se hizo con los *circuitos aritméticos convoluciones*. Este tipo de redes demostraron tener poder expresivo de la profundidad, que es como se denomina a que casi todas las funciones que pueden ser realizadas por una red profunda de tamaño polinómico, requiere un tamaño exponencial para ser realizable por una red superficial. Lo que el trabajo menciona como una descomposición tensorial generalizada es similar a una descomposición Tucker, pero utilizando un producto de tensores generalizado. Se demuestra que la red rectificadora convolucional es una universal en el caso de max *pooling*, pero no en el caso de uno promedio, y además que la eficiencia de profundidad es más débil para este tipo de redes que para los *circuitos aritméticos convoluciones*. Esto los lleva a los autores a concluir que este tipo de circuitos pueden llevar a arquitecturas que son superiores a las utilizadas actualmente.

En un trabajo posterior Khrulkov *et al.* [52] realiza el mismo ejercicio entre una red neuronal recurrente (RNN) con una descomposición TT [75] y además relaciona la profundidad de una red con el rango de la descomposición.

La conclusión es que se puede ampliar el resultado encontrado en el paper anterior [18] para las RNNs, ya que son exponencialmente más expresivas que las superficiales y además realiza un análisis comparativo entre el poder expresivo de las 3 redes mencionadas y su relación con redes de *deep learning*:

Descomposición	<i>deep learning</i>
Descomposición CP	Red superficial
Descomposición TT	RNN
Descomposición HT	CNN
Rango de la descomposición	Ancho de la red

Tabla 3: Correspondencia entre descomposición y redes de deep learning [52].

Se profundizará el análisis del trabajo de Khrulkov *et al.* en la sección donde se desarrolla la arquitectura propuesta (Sección 4.3), pero es importante analizar los resultados teóricos que expone. Se prueba el mismo teorema del poder expresivo de las redes para la descomposición TT que, para el caso de la descomposición HT, esto es que, dado un tensor d -dimensional de coeficientes aleatorios, descompuesto en el formato TT con rangos \mathbf{r} , este tensor va a tener con probabilidad 1 un rango exponencialmente más grande para una descomposición CP. Los resultados se pueden ver en esta tabla que se encuentran en el trabajo mencionado, dado una red de ancho r especificado en una columna, las filas corresponden con el límite superior en el ancho de la red equivalente de otro tipo, asumiendo que el tamaño de los *features maps* m es mayor al ancho de la red:

	Red TT	Red HT	Red CP
Red TT	r	$r^{\log_2(2)/2}$	r
Red HT	r^2	r	r
Red CP	$\geq r^{\frac{d}{2}}$	$\geq r^{\frac{d}{2}}$	r

Tabla 4: Comparación del poder expresivo de varias redes [52]. Dado un red de ancho r , especificada en una columna, la fila corresponde al límite superior de la profundidad de una red equivalente de otro tipo.

Existe otro trabajo de los mismos autores, Khrulkov *et al.* 2019[51], que también será mencionado posteriormente a la hora de desarrollar la arquitectura propuesta. Continuando con el hilo de los trabajos tanto de los mismos autores, como los escritos por Cohen *et al.* [18] [20], se demuestra que redes RNN que utilizan alguna no linealidad, como ReLU, también son universales y tienen “eficiencia de profundidad”. Esto se demuestra esto definiendo un producto tensorial generalizado para poder construir una red superficial y una red RNN, ambas “generalizadas” que utilizan no linealidades y luego se utilizan estas redes para sus demostraciones.

4.3. Arquitectura propuesta: Tensor Ring en deep learning

Tomando como base que la información de un tensor multidimensional de gran tamaño puede ser almacenada de manera eficiente y compacta en una descomposición tensorial sin demasiada pérdida,

en esta tesis, se propone utilizar *Tensor Ring* como arquitectura de *deep learning* y compararla con la solución basada en TT estudiada en la literatura [18, 51, 52]. La motivación de usar *Tensor Ring*, es que se ha demostrado que puede tener mayor poder de compresión comparada con la *Tensor Train* [98].

La red desarrollada es una red neuronal recurrente o RNN (ver sección 2.3). Este tipo de redes se caracterizan por tener conexiones que la retroalimentan. Así como fue mencionado antes, se suelen utilizar para procesar datos secuenciales, y en la sección 5.1 se detalla en profundidad qué bases de datos se usaron para probar esta arquitectura. La red RNN, como se especificó anteriormente (ver ecuación (Ec. 2.9)) se modela con un estado oculto $\mathbf{h}^{(t)}$ en el tiempo t , con una entrada $\mathbf{x}^{(t)}$ y un grupo de parámetros θ . Algo característico de las redes RNN es que comparten parámetros a través del modelo. Esto le permite aprender a generalizar de una manera más rápida y mejor.

Asumimos que todas las entradas de cualquiera de las bases de datos utilizadas pueden ser escritas como:

$$\mathbf{X} = \left(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(\tau)} \right), \mathbf{x}^{(t)} \in \mathbb{R}^n. \quad (\text{Ec. 4.4})$$

Éste es el caso para una única entrada, luego generalizaremos las ecuaciones para múltiples entradas, haciendo posible que sea utilizable con *batches* de entradas. Esto es muy importante debido a que, sin esa capacidad, el aprendizaje de la red puede complicarse. Otra observación importante es que la entrada de la red está dividida en secuencias de tamaño variable, donde cada uno de los índices t representa la posición de ese fragmento dentro de la secuencia y no necesariamente indica un paso en el tiempo.

Tomando como ejemplo algunos de los trabajos anteriores [51, 52], se decidió agregar un *feature map* para preprocesar los datos antes de que ser ingresados a la red, éste hace una transformación $\mathbf{f}_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^m$ y tiene la forma:

$$\mathbf{f}_\theta(\mathbf{x}) = \theta(\mathbf{A}\mathbf{x} + \mathbf{b}), \quad (\text{Ec. 4.5})$$

siendo θ una función de activación ReLU (Ec. 2.2), $\mathbf{A} \in \mathbb{R}^{m \times n}$ una matriz de transformación lineal y $\mathbf{b} \in \mathbb{R}^m$ un vector de *bias*. Su principal función es poder atrapar los *features* más importantes de la entrada a través de una transformación, esto puede ocasionar que sus dimensiones sean modificadas, dependiendo de que tan densa sea la información y con qué cantidad de *features* m se pueda lograr una buena representación. Este hiperparámetro m resulta muy importante y se abordará con mayor profundidad más adelante. Afecta directamente tanto la precisión de la clasificación, como la cantidad de parámetros de la red. En el caso de utilizarla para imágenes esta transformación se asemeja bastante a una transformación de una convolución tradicional, donde cada uno de los fragmentos es proyectado por la misma con parámetros compartidos (\mathbf{A} y \mathbf{b}) por todos los fragmentos, seguida de una función de activación (θ).

Una *score function* ℓ^y de una clase y representa numéricamente que tan probable es la clase a ser la clase de la entrada \mathbf{X} , ésta luego es usada para alimentar a la función *softmax* que luego se utiliza para calcular la pérdida (ver sección 2.1). Este tipo de estrategia llamada “one-versus-all”, suele ser utilizada para la clasificación de datos con clases pre-asignadas ocultas. Las *score functions* utilizadas en los trabajos de Cohen *et al.* 2016 [18] y Stoudenmire & Schwab 2016 [87] que fueron adoptadas por los trabajos posteriores pueden ser escritas de la forma:

$$\ell^y(\mathbf{X}) = \left\langle \mathcal{W}^{(y)}, \Phi(\mathbf{X}) \right\rangle \quad (\text{Ec. 4.6})$$

Siendo $\Phi(\mathbf{X}) \in \mathbb{R}^{m \times m \times \dots \times m}$ un *feature tensor* y $\mathcal{W}^{(y)} \in \mathbb{R}^{m \times m \times \dots \times m}$ un tensor de pesos entrenables, ambos tienen τ dimensiones. Donde $\Phi(\mathbf{X})$ es un tensor de rango 1 y está definido como:

$$\Phi(\mathbf{X}) = \mathbf{f}_\theta(\mathbf{x}^{(1)}) \otimes \mathbf{f}_\theta(\mathbf{x}^{(2)}) \otimes \dots \otimes \mathbf{f}_\theta(\mathbf{x}^{(\tau)}) \quad (\text{Ec. 4.7})$$

El tensor $\Phi(\mathbf{X})$ es un producto de un conjunto de *feature maps* locales $f_\theta(\mathbf{x}^{(j)})$ aplicada a cada fragmento $\mathbf{x}^{(j)}$ de la entrada \mathbf{X} . Se podría aplicar un *feature map* local diferente para cada componente $\mathbf{x}^{(j)}$, cada uno con una dimensión diferente, pero se utilizó una única transformación (Ec. 4.5) para todos, considerando las entradas homogéneas. En este caso se utiliza la función definida anteriormente (Ec. 4.5) pero en otros papers similares como *Stoudenmire & Schwab 2016* [87] se han utilizado funciones como seno y coseno, como una métrica de distancia que agrupe las diferentes entradas. Una representación del tensor $\Phi(\mathbf{X})$ resultante sería:

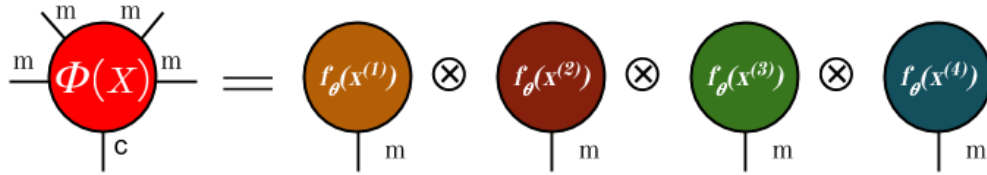


Figura 20: Ejemplo de un *feature tensor* de rango-1 compuesto por el producto de un conjunto de vectores *feature maps* locales $f_\theta(\mathbf{x}^{(j)})$ aplicados a cada fragmento $\mathbf{x}^{(j)}$.

Una *score function* (Ec. 4.6) para una clase particular ($y = 1$) se puede representar de la siguiente manera:

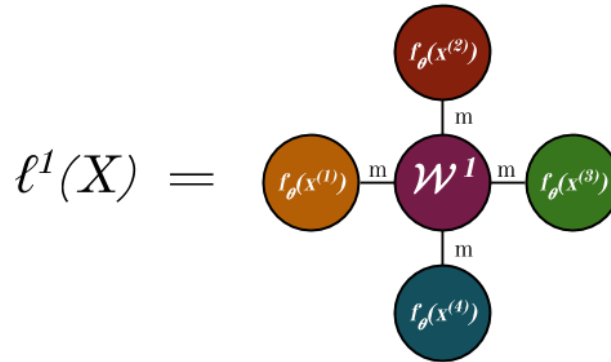


Figura 21: Ejemplo de una *score function* $\ell^y(\mathbf{X})$ (Ec. 4.6) para la clase $y = 1$.

El tensor $\mathcal{W}^{(y)}$ es un tensor que requiere de una cantidad exponencial de memoria para ser almacenado, la cantidad de parámetros es de $\mathcal{O}(m^\tau)$. Si se agrupa un tensor por cada clase y , suponiendo una cantidad de clases c , la cantidad de parámetros total es de $\mathcal{O}(c \cdot m^\tau)$. Se agrupan todos los tensores $\mathcal{W}^{(y)}$ formando un tensor de pesos entrenables, que es el componente principal de la red $\mathcal{W} \in \mathbb{R}^{c \times m \times m \times \dots \times m}$ (ver Figura 22). Siendo el elemento $\mathcal{W}_{y i_1 i_2 \dots i_N}$ igual al elemento $\mathcal{W}_{i_1 i_2 \dots i_N}^{(y)}$ del tensor original. El tensor de pesos entrenables \mathcal{W} resultante sería:

$$\mathcal{W} = (\mathcal{W}^{(1)}, \mathcal{W}^{(2)}, \dots, \mathcal{W}^{(c)}) \quad (\text{Ec. 4.8})$$

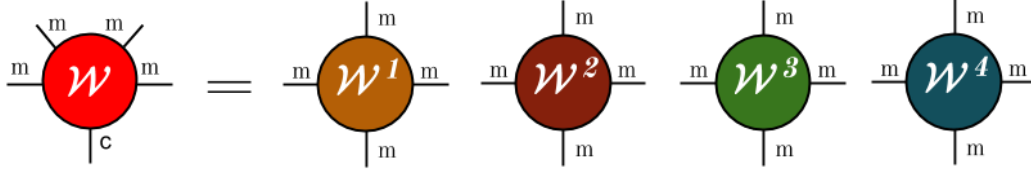


Figura 22: Ejemplo de un tensor \mathcal{W} de pesos entrenables, con $c = 4$, compuesto por un conjunto de tensores de pesos entrenables aplicados a cada fragmento $\mathbf{x}^{(j)}$.

Finalmente se puede escribir un vector de *score functions* tiene la fórmula:

$$\ell(\mathbf{X}) = \left(\left\langle \mathcal{W}^{(1)}, \Phi(\mathbf{X}) \right\rangle, \left\langle \mathcal{W}^{(2)}, \Phi(\mathbf{X}) \right\rangle, \dots, \left\langle \mathcal{W}^{(c)}, \Phi(\mathbf{X}) \right\rangle \right) \quad (\text{Ec. 4.9})$$

$$\ell(\mathbf{X}) = \left(\ell^{(1)}(\mathbf{X}), \ell^{(2)}(\mathbf{X}), \dots, \ell^{(c)}(\mathbf{X}) \right) \quad (\text{Ec. 4.10})$$

Dado esto se define $\ell^{(y)}$ manera:

$$\ell^{(y)}(\mathbf{X}) = \sum_{i_1=1}^{I_1} \sum_{i_2=1}^{I_2} \dots \sum_{i_N=1}^{I_N} \mathcal{W}_{i_1 i_2 \dots i_N}^{(y)} \cdot \Phi(\mathbf{X})_{i_1 i_2 \dots i_N} \quad (\text{Ec. 4.11})$$

El vector de *score functions* $\ell(\mathbf{X})$ es aquel que luego será alimentado a la función *softmax*. El problema es que, como mencionamos anteriormente, el espacio para almacenar \mathcal{W} es exponencial, lo que implica un problema por la cantidad de memoria y operaciones necesarias para realizar el cálculo de $\ell(\mathbf{X})$ (Ec. 4.11). Aquí se presenta la oportunidad de aplicar una descomposición tensorial, la cual nos permite dar una representación más compacta del tensor de pesos entrenables, reduciendo así tanto la necesidad de memoria como la cantidad de operaciones.

4.3.1. La nueva arquitectura es compatible con una RNN

Este trabajo se propone implementar una descomposición *Tensor Ring* sobre el tensor \mathcal{W} , además convirtiéndose a través de la arquitectura implementada en una red recurrente. Para demostrar la relación entre una red RNN y la descomposición *Tensor Ring* se utiliza un conjunto de pasos parecidos a los usados en [18, 52, 87]. Se puede empezar reescribiendo la ecuación (Ec. 3.8) para una descomposición TR de un tensor $\mathcal{W} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_d}$ de orden d como:

$$\mathcal{W} = \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_d=1}^{R_d} \mathbf{g}^{(1)}(r_1, r_2) \otimes \mathbf{g}^{(2)}(r_2, r_3) \otimes \dots \otimes \mathbf{g}^{(d)}(r_d, r_1), \quad (\text{Ec. 4.12})$$

donde, continuando con las convenciones establecidas anteriormente (ver sección 3.4), $\mathbf{g}^{(k)}(r_k, r_{k+1})$ es una *fiber* del tensor núcleo $\mathcal{G}^{(k)} \in \mathbb{R}^{R_k \times N_k \times R_{k+1}}$ de orden 3 (ver Figura 11), todos juntos éstos conforman la descomposición tensorial. La ecuación (Ec. 4.12) indica que todo el tensor puede ser generado por la suma de tensores de rango 1 formados por d vectores tomados de cada núcleo. Esta última afirmación permite deducir que la descomposición CPD (Ec. 3.2) es un caso especial. Otra observación interesante es que una descomposición *Tensor Train* también se puede escribir con la ecuación (Ec. 4.12) pero con la restricción $R_1 = 1$.

La fórmula (Ec. 4.12) a su vez se puede reescribir en término de los *slices* de los tensores núcleos $\mathcal{G}^{(k)}(n_k) \in \mathbb{R}^{R_k \times R_{k+1}}$ utilizando sus elementos $g^{(k)}(n_k)_{r_k, r_{k+1}}$, interpretándolo como la traza de un

producto secuencial de esas matrices:

$$\begin{aligned}
 w_{r_1, r_2, \dots, r_d} &= \text{Tr}\left\{\prod_{k=1}^d G^{(k)}(i_k)\right\} = \text{Tr}\left\{\prod_{k=1}^d G^{(1)}(i_1)G^{(2)}(i_2)\dots G^{(d)}(i_d)\right\} \\
 &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_d=1}^{R_d} \prod_{k=1}^d g^{(k)}(n_k)_{r_k, r_{k+1}}
 \end{aligned} \tag{Ec. 4.13}$$

Suponiendo que $\mathcal{W} \in \mathbb{R}^{M \times M \times \dots \times M \times \tau}$ e un tensor de pesos entrenables, introduciendo la ecuación (Ec. 4.13) en (Ec. 4.6) la ecuación queda como:

$$\begin{aligned}
 \ell^y(\mathbf{X}) &= \left\langle \mathcal{W}^{(y)}, \Phi(\mathbf{X}) \right\rangle \\
 &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_d=1}^{R_d} \prod_{t=1}^{\tau} \langle \mathbf{f}_{\theta}(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle
 \end{aligned} \tag{Ec. 4.14}$$

La ecuación (Ec. 4.13) nos permite escribir una *score function* $\ell^y(\mathbf{X})$ de una clase y para una entrada $\mathbf{X} \in \mathbb{R}^{\tau \times N}$ como un producto entre la descomposición TR del tensor $\mathcal{W}^{(y)}$ de pesos entrenables y los *features maps* $\mathbf{f}_{\theta}(\mathbf{x}^{(t)})$. Este desarrollo es para una sola clase, pero vale la pena remarcar la diferencia de memoria necesaria comparándola con la arquitectura clásica (Ec. 4.6). Como se mencionó anteriormente para almacenar tanto en el caso de $\mathcal{W}^{(y)}$ como para $\Phi(\mathbf{X})$ requieren en el orden de $\mathcal{O}(m^{\tau})$ cada uno, de orden exponencial. Con la nueva arquitectura se requiere de $\mathcal{O}(\tau \cdot m \cdot R^2)$, siendo R el rango más grande de los tensores núcleos, para almacenar los tensores núcleos de la descomposición de $\mathcal{W}^{(y)}$, además se requiere de $\mathcal{O}(\tau \cdot m)$ para almacenar todos los *features maps* $\mathbf{f}_{\theta}(\mathbf{x}^{(t)})$. El orden del almacenamiento en vez de ser exponencial pasa a ser polinómico.

Para terminar de conectar la descomposición con la red, es importante mostrar que la estructura de la *score function* bajo la nueva arquitectura de la descomposición TR es recurrente, lo que permite darle la etiqueta de una red RNN. Se define al primer estado oculto para un tiempo $t = 1$, cada elemento $h_{r_1, r_2}^{(1)}$ de la matriz $\mathbf{H}^{(1)} \in \mathbb{R}^{R_1 \times R_2}$ como:

$$h_{r_1, r_2}^{(1)} = \langle \mathbf{f}_{\theta}(\mathbf{x}^{(1)}), \mathbf{g}^{(1)}(r_1, r_2) \rangle \tag{Ec. 4.15}$$

El siguiente paso es definir un estado oculto $\mathbf{H}^{(t)} \in \mathbb{R}^{R_1, R_{t+1}}$ para un tiempo $t = 2, 3, \dots, \tau$, en este caso se introduce el estado oculto anterior en el cálculo:

$$h_{r_t, r_{t+1}}^{(t)} = \sum_{r_t=1}^{R_t} \langle \mathbf{f}_{\theta}(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot h_{r_{t-1}, r_t}^{(t-1)} \quad \forall t = 2, 3, \dots, \tau \tag{Ec. 4.16}$$

Para calcular el último estado oculto es importante recordar que el último tensor de la descomposición tiene la forma de $\mathcal{G}^{(\tau)} \in \mathbb{R}^{R_{\tau} \times N_{\tau} \times R_{\tau+1}}$ teniendo en cuenta que $R_{\tau+1} = R_1$ (ver Sección 3.4). Esto tiene como consecuencia directa que el estado $\mathbf{H}^{(\tau)} \in \mathbb{R}^{R_1, R_{\tau+1}}$ es una matriz cuadrada, lo cual permite la escritura del último estado oculto $h^{(\tau+1)} \in \mathbb{R}^1$ como la traza:

$$h^{(\tau+1)} = \sum_{r_1=1}^{R_1} h_{r_1, r_1}^{(\tau)} = \text{Tr}(\mathbf{H}^{(\tau)}) \tag{Ec. 4.17}$$

Tomando en cuenta el cálculo de la *score function* para una clase escrita como una descomposición de *Tensor Ring* (Ec. 4.14) ésta se puede reescribir en función de los estados ocultos (Ec. 4.15) (Ec. 4.16) (Ec. 4.17) como:

$$\ell^y(\mathbf{X}) = h^{(\tau+1)} \tag{Ec. 4.18}$$

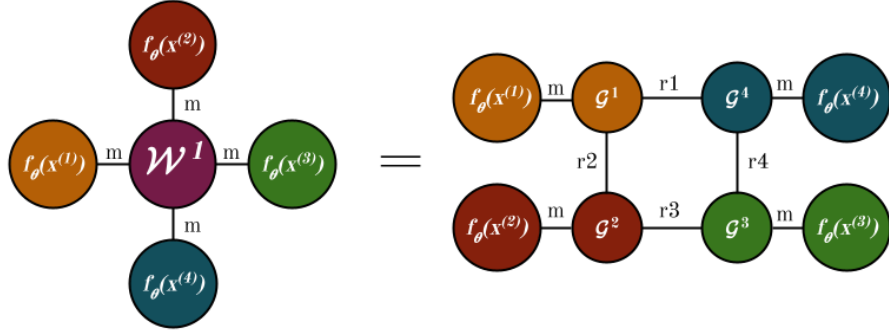


Figura 23: Ejemplo de una *score function* (Ec. 4.6) de orden descompuesta con una descomposición *Tensor Ring* (Ec. 4.18).

Si en vez de escribir las ecuaciones de estado en función de *fibers*, se expresan directamente como producto de los elementos del tensor, éstas (Ec. 4.15) (Ec. 4.16) podrían ser reescritas como:

$$\mathbf{h}_{r_1, r_2}^{(1)} = \sum_{i=1}^m g_{r_1, i, r_2}^{(1)} \cdot f_{\theta}(\mathbf{x}^{(1)})_i \quad (\text{Ec. 4.19})$$

$$h_{r_t, r_{t+1}}^{(t)} = \sum_{r_t=1}^{R_t} \sum_{i=1}^m g_{r_t, i, r_{t+1}}^{(t)} \cdot f_{\theta}(\mathbf{x}^{(t)})_i \cdot h_{r_{t-1}, r_t}^{(1)} \quad \forall t = 2, 3, \dots, \tau \quad (\text{Ec. 4.20})$$

Escrito como producto de los tensores núcleos con vectores y matrices, las ecuaciones anteriores se pueden reescribir como:

$$\mathbf{H}^{(1)} = \mathbf{f}_{\theta}(\mathbf{x}^{(1)})^{\top} \mathcal{G}^{(1)} \quad (\text{Ec. 4.21})$$

$$\mathbf{H}^{(t)} = \mathbf{f}_{\theta}(\mathbf{x}^{(t)})^{\top} \mathcal{G}^{(t)} \mathbf{H}^{(t-1)} \quad \forall t = 2, 3, \dots, \tau \quad (\text{Ec. 4.22})$$

Podemos además, utilizando una matriz auxiliar que represente un estado inicial $\mathbf{H}^0 \in \mathbb{R}^{R_1 \times R_1}$, calcular todos los estados (excepto el de salida) con la ecuación (Ec. 4.22). Este atajo de introducir un estado inicial es bastante común en las implementaciones de redes RNN.

4.3.2. Cómo aplicar el modelo a un conjunto de entradas (*batch*)

Es notable que el desarrollo que se introdujo en la ecuación anterior (Ec. 4.22) sea válido para entradas con un *batch* de tamaño 1 para una sola clase y . Para poder generalizarlo hay que modificar cómo se realiza la operación del *feature map* (Ec. 4.5), en la sección de implementación 5.1 se profundiza cómo la operación es realizada con un *batch* de entrada. La fórmula se modifica debido a que ahora tenemos una entrada de $\mathbf{X}^{(t)} \in \mathbb{R}^{B \times n}$ siendo $B \in \mathbb{R}$ el tamaño del *batch*, utilizando los mismos parámetros que en la fórmula original, $\mathbf{A} \in \mathbb{R}^{m \times n}$ una matriz de transformación lineal y $\mathbf{b} \in \mathbb{R}^m$ un vector de *bias*, definen un *feature map* $\mathbf{F}_{\theta}(\mathbf{X}^{(t)}) \in \mathbb{R}^{B \times m}$ que se calcula:

$$\mathbf{F}_{\theta}(\mathbf{X}^{(t)}) = \theta(\mathbf{A}\mathbf{X}^{(t)} + \mathbf{b}) \quad (\text{Ec. 4.23})$$

Es relevante mencionar que la función de activación continúa siendo una ReLU (Ec. 2.2) y que la operación de suma del vector de *bias* se realiza sobre cada columna de la matriz resultante del producto. Finalmente el resultado de la fórmula (Ec. 4.23) es una matriz, lo que termina generando que se modifique el estado oculto de la red para que pase a ser un tensor. Éste contendrá los

estados de la misma manera que lo hacía anteriormente, pero se le agregará una dimensión de *batch*, $\mathcal{H}^{(t)}$ es un estado oculto definido $\mathcal{H}^{(t)} \in \mathbb{R}^{R_0 \times B \times R_t}$. Utilizando la nueva definición del *feature map* (Ec. 4.23), planteamos el cálculo del estado oculto (Ec. 4.16) pero esta vez como un tensor, cada elemento se calcula como:

$$h_{r_0, b, r_{t+1}}^{(t)} = \sum_{r_t=1}^{R_t} \sum_{i=1}^m g_{r_t, i, r_{t+1}}^{(t)} \cdot f_{\theta}(\mathbf{X}^{(t)})_{b, i} \cdot h_{r_0, b, r_t}^{(t-1)} \quad (\text{Ec. 4.24})$$

Comparando la última expresión (Ec. 4.24) con la misma sin *batches* (Ec. 4.20), se puede ver que la expresión es en esencia la misma, sólo que a través de una dimensión más. Se puede obtener la fórmula generalizada como un producto de vectores, análoga a (Ec. 4.22) como:

$$\mathcal{H}^{(t)} = \mathbf{F}_{\theta}(\mathbf{X}^{(t)}) \mathcal{G}^{(t)} \mathcal{H}^{(t-1)} \quad (\text{Ec. 4.25})$$

Para obtener el último estado oculto $\mathbf{h}^{(\tau+1)} \in \mathbb{R}^B$ hay que realizar la operación de traza a través de la dimensión de *batch*:

$$h_b^{(\tau+1)} = \text{Tr}(\mathbf{H}^{(\tau)}(b)) \quad (\text{Ec. 4.26})$$

El score function se calcula en base a la entrada $\mathcal{X} \in \mathbb{R}^{B \times \tau \times n}$, que es dividida en τ divisiones $\mathbf{X}^{(t)} \in \mathbb{R}^{B \times n} \forall t = 1, 2, \dots, \tau$. La expresión actualizada queda definida:

$$\ell^y(\mathbf{X}) = \mathbf{h}^{(\tau+1)} \quad (\text{Ec. 4.27})$$

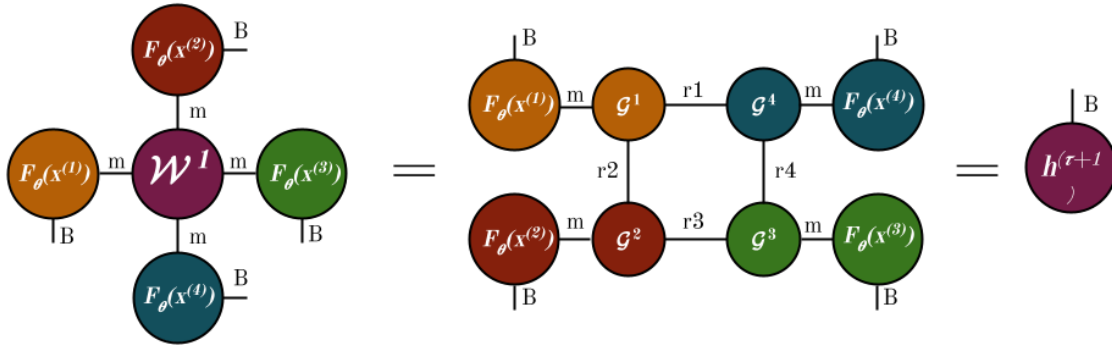


Figura 24: Ejemplo de una *score function* (Ec. 4.6) de orden descompuesta con una descomposición *Tensor Ring* con una dimensión de *batch* (Ec. 4.27).

4.3.3. Arquitecturas paralela, serial y compartida

Cabe destacar que los cálculos realizados en (Ec. 4.27) valen para una sola clase y , presentando un desafío en cómo plantear y elegir un modelo final para la arquitectura elegida. A la hora de proponer un modelo final, surgen varias alternativas a explorar, la primera alternativa, que es la más natural, es la que se denominó *paralela*. Ésta se basa en realizar una descomposición TR (ver Figura 24) para cada categoría $y = 1, 2, \dots, C$, donde cada clase tendrá τ tensores diferentes. La Figura 25 ilustra la arquitectura *paralela*.

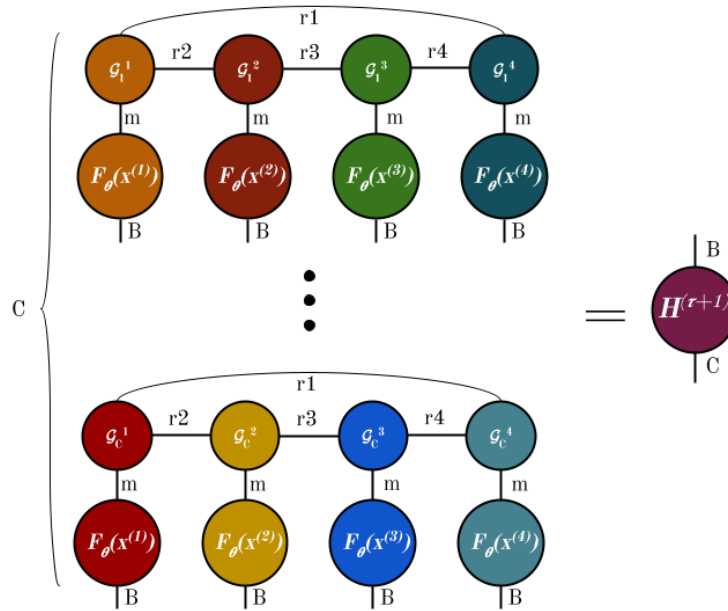


Figura 25: Ejemplo de una Arquitectura *Tensor Ring* paralela con $\tau = 4$.

El problema con este planteo es que multiplica por un factor de C , tanto a la cantidad de operaciones, como a la cantidad de parámetros de la red. Otro factor a tomar en cuenta es que este tipo de arquitectura no está *compartiendo* lo que aprende, en las diferentes clases. Más allá de que cada una tenga que clasificar una clase diferente, se puede argumentar que los tensores núcleos tienen mucho en común que aprender sobre las características de las entradas, ya transformadas por los *features map*. Esto lleva a un probable desperdicio de recursos y al procesamiento de muchas operaciones innecesarias, además del almacenamiento de información redundante. Este mal uso es conflictivo con el objetivo que se planteó al proponer esta arquitectura.

La siguiente propuesta generada fue la que se denominó *serial*, cuya idea es que existe la posibilidad compartir el conocimiento a través de los tensores núcleos de las diferentes clases. Este enfoque ya fue utilizado anteriormente en otros trabajos como el de *Stoudenmire & Schwab* 2016 [87]. Se experimentó utilizando una disposición donde, en vez de tener un tensor por categoría en cada paso, se tiene un núcleo por cada paso menos el primero, donde se mantiene la cantidad de tensores por categoría. Se eligió el primer paso por simplicidad, aunque se podría experimentar para determinar cual es la ubicación con la que se obtiene mejores resultados. Los tensores núcleos vinculados con la primer división pueden ser agrupados en un solo tensor núcleo de dimensiones $\mathcal{G}^{(1)} \in \mathbb{R}^{C \times R_1 \times m \times R_2}$. Este cambio no afecta la fórmula del estado oculto (Ec. 4.25), y podría ser representado como:

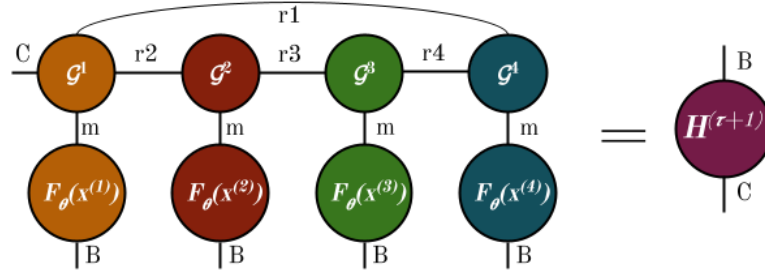


Figura 26: Ejemplo de Arquitectura *Tensor Ring* serial con $\tau = 4$. En este caso la dimensión de las categorías está en el primer tensor pero puede ser ubicada en cualquiera de ellos.

La arquitectura *serial* sigue sin tomar en cuenta el componente recurrente de la red: el de compartir parámetros a través de las diferentes divisiones. Esto permite generalizar el conocimiento a entradas que, tal vez, no están en el conjunto de entrenamiento, también utilizarlo cuando ciertas secuencias se repiten en diferentes posiciones de la entrada. La característica recurrente de las redes se puede ver en la ecuación genérica para el estado oculto de una red RNN (Ec. 2.9), se puede adaptar la fórmula expresada anteriormente (Ec. 4.25), para que tenga las mismas propiedades. La manera de lograr eso es igualar todos los tensores núcleos $\mathcal{G}^{(t)} = \mathcal{G}^{(t-1)}$ para todo $\mathcal{G}^{(t)} \in \mathbb{R}^{R_t \times m \times R_{t+1}}$ con $t = 2, \dots, \tau$. Imponiendo esta restricción permite compartir los parámetros a través de los pasos, habilitando la capacidad antes mencionada de aprender a través del tiempo. Para poder implementar esta estructura propuesta, se debe imponer un estado oculto inicial $\mathcal{H}^{(0)} \in \mathbb{R}^{R_{\tau+2} \times B \times R_1}$. Además es necesario agregar un componente más, al igual que en la arquitectura *serial*, para poder generalizar a todas las clases C , se puede utilizar un tensor $\mathcal{G}^{(\tau+1)} \in \mathbb{R}^{C \times R_{\tau+1} \times m \times R_{\tau+2}}$ que sería el último tensor del tren. Éste, al igual que el caso análogo, se puede pensar como C tensores diferentes, uno definido para cada clase. De la misma manera para este tensor núcleo se tiene una entrada con el mismo mecanismo de *padding* definido anteriormente, se genera artificialmente definida como $x^{(\tau+1)} \in \mathbb{R}^{B \times m}$. De esta manera se aprovecha al máximo el componente recurrente de la red, permitiéndole que aprenda de todos los pasos de la entrada. Una manera gráfica de representar esta nueva arquitectura que llamaremos *compartida* puede ser la siguiente:

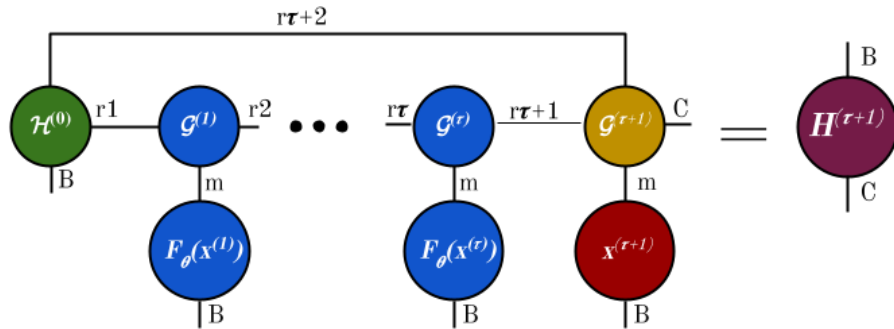


Figura 27: Arquitectura Tensor Ring *compartida*

Una observación interesante a notar es que la restricción de igualar todos los tensores también

fuerza que todos los rangos de la descomposición también sean iguales $R_1 = R_2 = \dots = R_{\tau+1}$.

4.4. Comparación con arquitecturas anteriores

4.4.1. Tensor Train (TT)

Como se menciono anteriormente, algunos trabajos ya se habían ocupado en implementar una red *Tensor Train*, el desarrollo es análogo al realizado en la sección 4.3. Este desarrollo se basó en el de Khrulkov *et al 2017* [52] y el trabajo del mismo equipo en 2019 [51] aunque tuvo algunas modificaciones para que sean comparables a la arquitectura propuesta.

Partiendo desde la misma fórmula de una *score function* (Ec. 4.6), se puede empezar por reescribir la ecuación (Ec. 3.7) para una descomposición TT de un tensor $\mathcal{T} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_d}$ de orden d como:

$$\mathcal{T} = \sum_{r_2=1}^{R_2} \dots \sum_{r_d=1}^{R_d} \mathbf{g}^{(1)}(r_1, r_2) \otimes \mathbf{g}^{(2)}(r_2, r_3) \otimes \dots \otimes \mathbf{g}^{(d)}(r_d, r_{d+1}) \quad (\text{Ec. 4.28})$$

Donde $\mathbf{g}^{(k)}(r_k, r_{k+1}) \in \mathbb{R}^{I_k}$ es una *fiber* del tensor núcleo $\mathcal{G}^{(k)} \in \mathbb{R}^{R_k \times I_k \times R_{k+1}}$ de orden 3. A diferencia de la descomposición de *Tensor Ring*, el primer y el último tensor son matrices $\mathbf{G}^{(1)} \in \mathbb{R}^{I_1 \times R_2}$ y $\mathbf{G}^{(d)} \in \mathbb{R}^{R_d \times I_d}$, por lo tanto, los rangos R_1 y R_{d+1} son igual a 1. Siguiendo con la comparación de ésta con la descomposición TR, si la vemos en comparación con la expresión anterior, escrita como la suma de un producto de *fibers*, se puede una vez más apreciar como *Tensor Train* es un caso especial donde se cumple la restricción $R_1 = 1$. Utilizando esta nueva fórmula con la antes mencionada podemos llegar a que la *score function* escrita en base a una descomposición TT del tensor \mathcal{W} se define como:

$$\begin{aligned} \ell^y(\mathbf{X}) &= \left\langle \mathcal{W}^{(y)}, \Phi(\mathbf{X}) \right\rangle \\ &= \sum_{r_2=1}^{R_2} \dots \sum_{r_d=1}^{R_d} \prod_{t=1}^{\tau} \sum_{n_t=1}^m \mathbf{f}_{\theta}(\mathbf{x}^{(t)})_{n_t} \cdot \mathbf{g}^{(t)}(n_t)_{r_t, r_{t+1}} \\ &= \sum_{r_2=1}^{R_2} \dots \sum_{r_d=1}^{R_d} \prod_{t=1}^{\tau} \langle \mathbf{f}_{\theta}(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \end{aligned} \quad (\text{Ec. 4.29})$$

La expresión (Ec. 4.29) nos permite escribir una *score función* $\ell^y(\mathbf{X})$ que tiene una entrada $\mathbf{X} \in \mathbb{R}^{\tau \times n}$, como un producto entre *fibers* $\mathbf{g}^{(t)}(r_t, r_{t+1})$ de los tensor núcleos de una descomposición *Tensor Ring* con los *feature map* $\mathbf{f}_{\theta}(\mathbf{x}^{(t)}) \in \mathbb{R}^m$, siendo $\mathbf{x}^{(t)} \in \mathbb{R}^n$ una de las divisiones de esa entrada. El orden de almacenamiento para guardar los núcleos de la descomposición es de $\mathcal{O}(\tau \cdot m \cdot R^2)$, siendo R el rango más grande de los tensores, además se requiere de $\mathcal{O}(\tau \cdot m)$ para almacenar los *feature map*. Comparándola con la arquitectura propuesta la necesidad de almacenamiento es la misma en orden y en ambos casos el orden de almacenamiento en vez de ser exponencial con el tensor y el feature map completo pasa a ser polinómico.

Al igual que en la arquitectura propuesta, es necesario mostrar que la *score function* para la descomposición de TT tiene una estructura recurrente para poder catalogarla como una red RNN. Lo primero es definir el estado oculto para un tiempo $t = 1$, teniendo en cuenta que el primer tensor núcleo $\mathbf{G}^{(1)} \in \mathbb{R}^{I_1 \times R_2}$ es una matriz, cada elemento $h_{r_2}^{(1)}$ del vector $\mathbf{h}^{(1)} \in \mathbb{R}^{R_2}$ como:

$$h_{r_2}^{(1)} = \langle \mathbf{f}_{\theta}(\mathbf{x}^{(1)}), \mathbf{g}^{(1)}(r_2) \rangle \quad (\text{Ec. 4.30})$$

A continuación podemos generalizar para el resto de los estados ocultos $\mathbf{h}^{(t)} \in \mathbb{R}^{R_t+1}$ para $t = 2, 3, \dots, \tau - 1$, definiendo cada elemento $h_{r_t}^{(t)}$ como:

$$h_{r_{t+1}}^{(t)} = \sum_{r_t=1}^{R_t} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot h_{r_t}^{(t-1)} \quad \forall t = 2, \dots, \tau \quad (\text{Ec. 4.31})$$

El estado final $h^{(\tau)} \in \mathbb{R}^1$ está definido en la ecuación anterior (Ec. 4.31), para realizar una equiparación entre la definición de este estado y el estado final de la arquitectura propuesta (Ec. 4.32), volvemos a escribirlo teniendo en cuenta que el tensor núcleo $\mathbf{G}^\tau \in \mathbb{R}^{R_\tau \times m}$ es una matriz:

$$h^{(\tau)} = \sum_{r_\tau=1}^{R_\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(\tau)}), \mathbf{g}^{(\tau)}(r_\tau, r_{\tau+1}) \rangle \cdot h_{r_\tau}^{(\tau-1)} \quad (\text{Ec. 4.32})$$

Finalmente una *score function* para una clase escrita como una descomposición *Tensor Train* (Ec. 4.29), usando las ecuaciones que definen los estados ocultos (Ec. 4.30), (Ec. 4.31), (Ec. 4.32) como:

$$\ell^y(\mathbf{X}) = h^{(\tau)} \quad (\text{Ec. 4.33})$$

La prueba de esta expresión (Ec. 4.33) se puede encontrar en el apéndice del trabajo de Khrulkov *et al.* 2019 [51], también se puede encontrar en el apéndice A (Ec. A.2), donde también se puede hallar la demostración (Ec. A.1) de la misma ecuación para la arquitectura de *Tensor Ring* (Ec. 4.18).

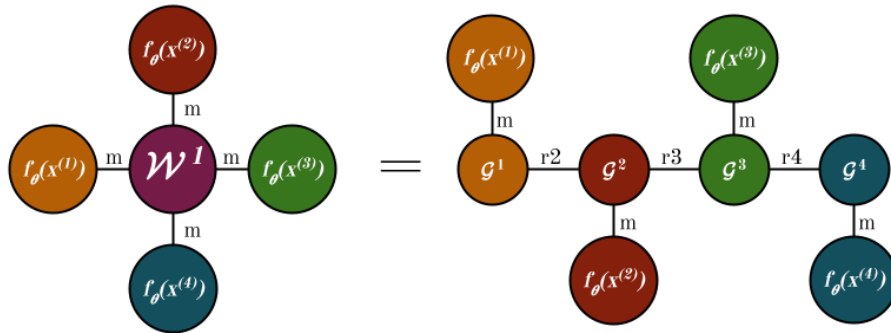


Figura 28: Ejemplo de una *score function* (Ec. 4.6) de orden descompuesta con una descomposición *Tensor Train* (Ec. 4.33).

Escribiendo las ecuaciones (Ec. 4.30), (Ec. 4.31) y (Ec. 4.32), respectivamente en vez de en función de *fibers*, directamente como producto de elementos de los tensores núcleos con los *feature tensor* se definen como:

$$h_{r_2}^{(1)} = \sum_{i=1}^m f_\theta(\mathbf{x}^{(1)})_i \cdot g_{i,r_2}^{(1)} \quad (\text{Ec. 4.34})$$

$$h_{r_{t+1}}^{(t)} = \sum_{r_t=1}^{R_t} \sum_{i=1}^m f_\theta(\mathbf{x}^{(t)})_i \cdot g_{r_t,i,r_{t+1}}^{(t)} \cdot h_{r_t}^{(t-1)} \quad \forall t = 2, 3, \dots, \tau \quad (\text{Ec. 4.35})$$

$$h^{(\tau)} = \sum_{r_\tau=1}^{R_\tau} \sum_{i=1}^m f_\theta(\mathbf{x}^{(\tau)})_i \cdot g_{i,r_\tau}^{(\tau)} \cdot h_{r_\tau}^{(\tau-1)} \quad (\text{Ec. 4.36})$$

Escrito como producto de los tensores núcleos con vectores y matrices, las ecuaciones anteriores se pueden reescribir como:

$$\mathbf{h}^{(1)} = \mathbf{f}_\theta(\mathbf{x}^{(1)})^\top \mathbf{G}^{(1)} \quad (\text{Ec. 4.37})$$

$$\mathbf{h}^{(t)} = \mathbf{f}_\theta(\mathbf{x}^{(t)})^\top \mathbf{G}^{(t)} \mathbf{h}^{(t-1)} \quad \forall t = 2, 3, \dots, \tau - 1 \quad (\text{Ec. 4.38})$$

$$\mathbf{h}^{(\tau)} = \mathbf{f}_\theta(\mathbf{x}^{(\tau)})^\top \mathbf{G}^{(\tau)} \mathbf{h}^{(\tau-1)} \quad (\text{Ec. 4.39})$$

Se pueden generalizar las 3 ecuaciones ((Ec. 4.37), (Ec. 4.38) y (Ec. 4.39)) utilizando solamente la ecuación (Ec. 4.38), planteando un estado inicial $\mathbf{h}^{(0)} \in \mathbb{R}^1$ e introduciéndolo en la ecuación (Ec. 4.37) para que cumpla con la fórmula de la expresión generalizada. Cuando comparamos la expresión generalizada (Ec. 4.38) con su análogo de la descomposición *Tensor Ring* (Ec. 4.22), es importante notar la diferencia entre la definición de los estados, una matriz $\mathbf{H}^{(t)} \in \mathbb{R}^{R_1, R_{t+1}} \forall t = 1, \dots, \tau$ para el caso de la descomposición TR y un vector $\mathbf{h}^{(t)} \in \mathbb{R}^{R_{t+1}} \forall t = 1, \dots, \tau$ para la descomposición *Tensor Train*. La desigualdad tanto en orden, como en tamaño, trae como consecuencia que haya una disparidad en la cantidad de operaciones necesarias para ambas arquitecturas. En el caso de la red TT podemos deducir que la operación generalizada es $\mathcal{O}(R^2 \cdot m)$, siendo R el rango más alto de todo el tensor. Si realizamos la operación para todos los τ pasos, este cálculo es del orden $\mathcal{O}(R^2 \cdot m \cdot \tau)$ para una entrada $\mathbf{X} \in \mathbb{R}^{\tau \times n}$. La operación de calcular los *feature map* de cada fragmento $\mathbf{x}^{(j)}$ de la entrada \mathbf{X} definida por la ecuación (Ec. 4.5) es del orden $\mathcal{O}(m \cdot n)$, por lo tanto el orden final de las operaciones de la red es de $\mathcal{O}(R^2 \cdot m \cdot \tau + n \cdot m \cdot \tau)$ o simplificado:

$$\mathcal{O}((R^2 + n) \cdot m \cdot \tau) \quad (\text{Ec. 4.40})$$

En el caso de la arquitectura basada en la descomposición *Tensor Ring*, el orden de la operación realizada en la expresión generalizada (Ec. 4.22) de cada paso es de $\mathcal{O}(R' \cdot R^2 \cdot m)$, el tener como estado oculto una matriz en lugar de un vector le agrega una dimensión más sobre la cual debe iterar, vale la pena notar que se define $R' = R_{\tau+2}$. Ésta es la misma sobre la cual se realiza la operación de traza y es la dimensión extra aludida anteriormente. De la misma manera que en la red mencionada anteriormente, se debe realizar esta operación para cada una de los τ pasos, el orden es de $\mathcal{O}(R' \cdot R^2 \cdot m \cdot \tau)$. La operación de traza realizada para calcular el último estado oculto (Ec. 4.17) es de $\mathcal{O}(R')$, respecto del orden del resto de las operaciones es despreciable, debido a que ésta es lineal comparada con el orden cúbico (siendo R del mismo orden que R') del resto de la red. Agregando también las operaciones que deben realizarse para calcular los *feature map* de cada fragmento $\mathbf{x}^{(j)}$ de la entrada \mathbf{X} , la cantidad total de operaciones de la red completa es:

$$\mathcal{O}((R' \cdot R^2 + n) \cdot m \cdot \tau) \quad (\text{Ec. 4.41})$$

Si se compara el orden deducido de la arquitectura con la descomposición TT (Ec. 4.40), podemos primero notar, ambas son iguales en el caso de que el rango $R' = R_{\tau+2} = 1$, que es un resultado trivial debido a que como mencionamos anteriormente la descomposición TR es una generalización de una descomposición TT, pero vale la pena mencionar que esta relación también se ve reflejada en el orden.

Al igual que en el caso de la arquitectura desarrollada para la descomposición TR, la ecuación (Ec. 4.38) generalizada se puede utilizar para entradas con *batch* de tamaño 1, y sólo clasifica de manera binaria con una sola clase y . Si se utiliza la operación de *feature map* con un *batch* de tamaño B (Ec. 4.23), teniendo como entrada un tensor $\mathcal{X} \in \mathbb{R}^{B \times n \times \tau}$ con B entradas distintas, cada entrada del *feature map* $\mathbf{F}_\theta(\mathbf{X}^{(t)}) \in \mathbb{R}^{B \times m}$ es un paso de la entrada $\mathbf{X}^{(t)} \in \mathbb{R}^{B \times n}$. Al igual que

con la arquitectura anterior, los parámetros del *feature map* no cambian, solamente se agrega esa dimensión al cálculo.

El estado oculto también sufrirá modificaciones cuando se le introduzca la dimensión del *batch*, éste quedará definido como $\mathbf{H}^{(t)} \in \mathbb{R}^{R_{t+2} \times B}$, se puede reescribir la ecuación que describe todos los elementos de un estado oculto para esta arquitectura (Ec. 4.35) utilizando la nueva expresión desarrollada para los *feature maps* (Ec. 4.23) como:

$$h_{r_{t+1},b}^{(t)} = \sum_{r_t=1}^{R_t} \sum_{i=1}^m f_{\theta}(\mathbf{x}^{(t)})_{b,i} \cdot g_{r_t,i,r_{t+1}}^{(t)} \cdot h_{r_t,b}^{(t-1)} \quad \forall t = 1, 2, \dots, \tau \quad (\text{Ec. 4.42})$$

Comparando la última expresión (Ec. 4.42) con la misma sin *batches* (Ec. 4.35) y sus análogas para la arquitectura propuesta (Ec. 4.20), (Ec. 4.24), podemos notar la simetría en las diferencias entre ellas, notando que no modifica en ninguno de ambos casos la cantidad de parámetros solamente como paraleliza las operaciones que deben realizarse, a diferencia de si el *batch* fuera de tamaño 1. La ecuación (Ec. 4.35) como un producto de tensores puede ser escrita como:

$$\mathbf{H}^{(t)} = \mathbf{F}_{\theta}(\mathbf{x}^{(t)})^{\top} \mathbf{G}^{(t)} \mathbf{H}^{(t-1)} \quad \forall t = 1, 2, \dots, \tau \quad (\text{Ec. 4.43})$$

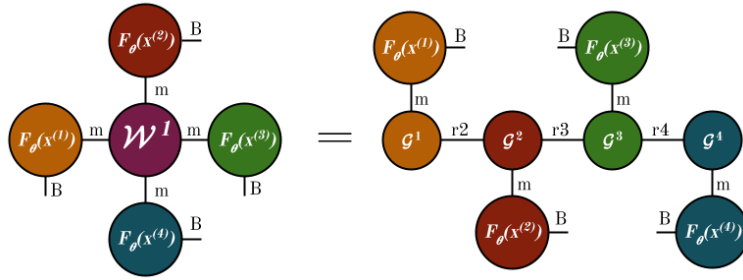
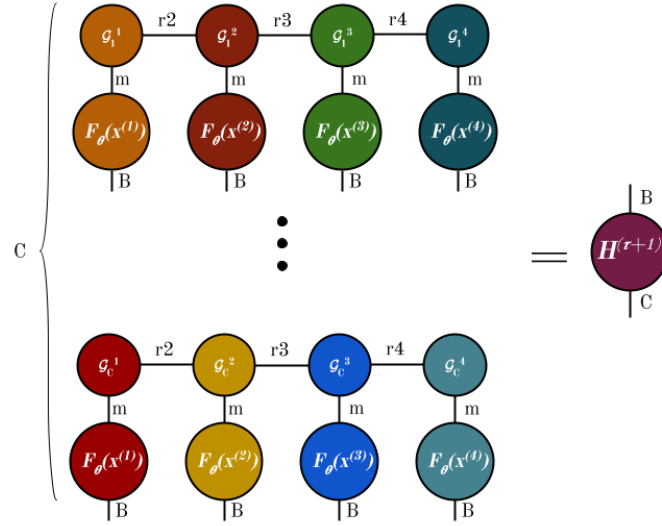


Figura 29: Ejemplo de una *score function* (Ec. 4.6) de orden descompuesta con una descomposición *Tensor Train* (Ec. 4.43).

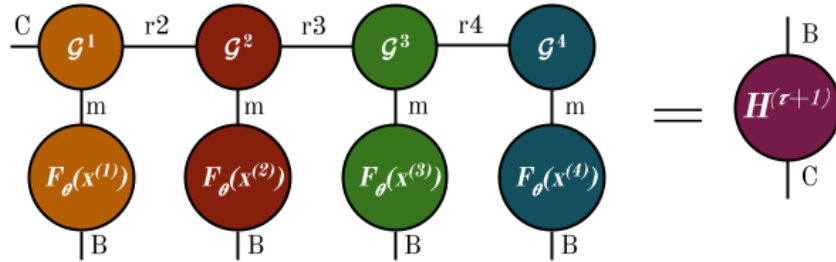
El último estado oculto $\mathbf{h}^{\tau} \in \mathbb{R}^B$ se puede calcular con la misma ecuación anterior (Ec. 4.43), teniendo en cuenta que el último tensor es una matriz $\mathbf{G}^{\tau} \in \mathbb{R}^{\tau \times m}$, éste se calcula:

$$\mathbf{h}^{\tau} = \mathbf{F}_{\theta}(\mathbf{x}^{(\tau)})^{\top} \mathbf{G}^{(\tau)} \mathbf{H}^{(\tau-1)} \quad (\text{Ec. 4.44})$$

Respecto del cálculo realizado en (Ec. 4.44) es importante recalcar que es válido para una única clase y , al igual que con la arquitectura de la descomposición de *Tensor Ring* se plantearon de manera las mismas alternativas para la asociada con la descomposición *Tensor Train*. La primera alternativa propuesta es la que se denominó *paralela*, donde se plantea una descomposición (ver Figura 29) por categoría para cada una, cada una estará compuesta de τ tensores núcleos. Una ilustración de la arquitectura sería la siguiente:


 Figura 30: Ejemplo de Arquitectura *Tensor Train* paralela con $\tau = 4$.

Esta arquitectura sufre los mismos problemas que la arquitectura *Tensor Ring* paralela, padeciendo de una redundancia tanto en parámetros como en operaciones, debido a que sufren de la incapacidad de compartir el conocimiento a través de las clases. Debido a estas debilidades se planteó también para una arquitectura *serial*, en este caso al igual que el análogo para la otra arquitectura, comparten los tensores para todas las clases pero para el caso de uno de los pasos, el primero, las clases están paralelizadas, teniendo un tensor para cada clase. Los tensores núcleos vinculados con el primer paso pueden ser agrupados en un solo tensor núcleo de dimensiones $\mathcal{G}^{(1)} \in \mathbb{R}^{C \times m \times R_2}$. La ilustración de la arquitectura serial puede ser representada como:


 Figura 31: Ejemplo de Arquitectura *Tensor Train* serial con $\tau = 4$. En este caso la dimensión de las categorías está en el primer tensor pero puede ser ubicados en cualquiera de ellos.

Finalmente se planteó la arquitectura llamada *compartida*, al igual que para el caso de *Tensor Ring*, debido a que como se mencionó antes, la arquitectura *serial* carece del componente recurrente que tienen las redes RNN. Se establece un estado oculto inicial $\mathbf{H}^{(0)} \in \mathbb{R}^{R_B \times R_0}$, a diferencia del caso planteado anteriormente, no tiene el rango extra que en el caso de *Tensor Ring* es el que permite realizar la operación de traza, y por eso se podría interpretar la descomposición como un caso especial de la anterior pero siendo el último rango de valor $R_{\tau+2} = 1$. A diferencia del ejemplo

anterior, es este caso se planteó un tensor inicial, $\mathcal{G}^{(0)} \in \mathbb{R}^{R_0 \times m \times R_1}$, que actúa como generador el estado inicial para el tensor que actúa como el núcleo de la red, recibe el estado oculto y una entrada, $\mathbf{X}^{(0)} \in \mathbb{R}^{B \times m}$. Para este trabajo definiremos el valor de $R_0 = 1$ para todos los experimentos, con el mismo mecanismo de *padding* descrito anteriormente, una manera de interpretar la primera operación como si estuviera aprendiendo el estado inicial óptimo $\mathbf{H}^{(0)} \in \mathbb{R}^{R_B \times R_1}$:

$$h_{r_1,b}^{(0)} = \sum_{i=1}^m x_{b,i}^{(0)} \cdot g_{r_0,i,r_1}^{(0)} \cdot h_{r_0,b}'^{(0)} \quad (\text{Ec. 4.45})$$

El componente recurrente de la red puede considerarse que es el mismo tensor núcleo $\mathcal{G}^{(t)} = \mathcal{G}^{(t-1)}$ para todo $\mathcal{G}^{(t)} \in \mathbb{R}^{R_t \times m \times R_{t+1}}$ con $t = 2, \dots, \tau$. Esto permite compartir los parámetros a través de los pasos, habilitando la capacidad antes mencionada de aprender a través del tiempo. Además, al igual que en la arquitectura *serial*, para poder generalizar a todas las clases C , se puede utilizar un tensor $\mathcal{G}^{(\tau+1)} \in \mathbb{R}^{C \times R_{\tau+1} \times m \times R_{\tau+2}}$ que es el último tensor del tren. Pero en este caso, como se mencionó anteriormente, el último rango es igual a uno, esto cambia de manera circunstancial el orden interno de las operaciones de la red así como el de su estado, a través de las iteraciones sobre el tensor núcleo. Tomando en cuenta esto, la arquitectura *compartida* se puede ilustrar de la siguiente manera:

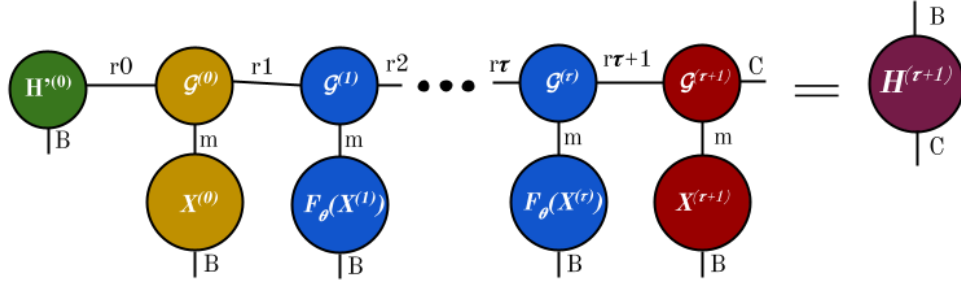


Figura 32: Arquitectura *Tensor Train* compartida.

Dados estos cambios en la arquitectura se puede volver a calcular el orden de operaciones necesarios para el caso de la arquitectura *compartida* para ambas descomposiciones. Para el caso de la arquitectura con la descomposición *Tensor Train*, el orden calculado anteriormente (Ec. 4.40) se modifica con la adición del cálculo por *batch*, con un *batch* de tamaño B , además de agregar el último tensor, sumando un paso más, que funciona como C tensores en paralelo, uno por cada categoría para las C categorías. La primera diferencia importante a notar es que la modificación en el cálculo de operaciones de los *features maps* $\mathbf{F}_{\theta}(\mathbf{X}^{(t)}) \in \mathbb{R}^{B \times m}$ (Ec. 4.23) para cada una de las entradas $\mathbf{X}^{(t)} \in \mathbb{R}^{B \times n}$ para cada paso $t = 1, \dots, \tau$, el orden se vuelve $\mathcal{O}(n \cdot m \cdot \tau \cdot B)$. El estado oculto definido como $\mathbf{H}^{(t)} \in \mathbb{R}^{R_{t+2} \times B}$ (Ec. 4.43) para los primeros τ pasos, está definido de manera idéntica que el utilizado para calcular el orden anterior (Ec. 4.38) pero haciendo el cálculo para cada elemento del *batch*, utilizando la ecuación por elemento (Ec. 4.42), podemos deducir el orden como $\mathcal{O}(R^2 \cdot m \cdot B)$ para cada paso t , para todo $t = 1, \dots, \tau$. También hay que tener en cuenta que el cálculo es el mismo para el último tensor, pero se realiza la misma para cada una de las C categorías, la expresión para cada uno de los elementos del último estado $\mathbf{H}^{(\tau+1)} \in \mathbb{R}^{C \times B}$ se puede definir como:

$$h_{c,b}^{(\tau+1)} = \sum_{r_{\tau}=1}^{R_{\tau}} \sum_{i=1}^m x_{b,i}^{(\tau+1)} \cdot g_{r_{\tau},i,c}^{(\tau+1)} \cdot h_{r_{\tau},b}^{(\tau)} \quad (\text{Ec. 4.46})$$

De la expresión anterior (Ec. 4.46) se puede deducir que el orden de esta operación en particular es de $\mathcal{O}(R \cdot m \cdot C \cdot B)$ y la descrita por la ecuación (Ec. 4.45) teniendo en cuenta que se definió $R_0 = 1$ de orden $\mathcal{O}(R \cdot m \cdot B)$, observando el resto de los ordenes calculados, el orden total para la arquitectura compartida de la descomposición de *Tensor Train* es:

$$\mathcal{O}(((R^2 + n) \cdot \tau + R \cdot C + R)) \cdot m \cdot B \quad (\text{Ec. 4.47})$$

Para calcular el orden de la arquitectura compartida de la descomposición *Tensor Ring* y poder compararla con el de la otra descomposición, hay que notar que el orden del cálculo de los *features maps* $\mathbf{F}_\theta(\mathbf{X}^{(t)}) \in \mathbb{R}^{B \times m}$ (Ec. 4.23) es el mismo que para el caso anterior $\mathcal{O}(n \cdot m \cdot \tau \cdot B)$. Para poder determinar el orden de los cálculos de los primeros τ pasos, se debe primero observar la expresión generalizada (Ec. 4.24), teniendo en cuenta ésta se puede determinar que el orden es $\mathcal{O}(R' \cdot R^2 \cdot m \cdot \tau \cdot B)$. Finalmente se necesita calcular el último paso $t = \tau + 1$, donde el último tensor realiza la misma operación con la misma expresión generalizada (Ec. 4.24) mencionada anteriormente para cada uno de las C categorías. La expresión para cada uno de los elemento del estado $\mathcal{H}^{(\tau+1)} \in \mathbb{R}^{R_0 \times B \times R_{\tau+2} \times C}$ se puede definir como:

$$h_{r_0, b, r_{\tau+2}, c}^{(\tau+1)} = \sum_{r_{\tau+1}=1}^{R_{\tau+1}} \sum_{i=1}^m x_{b,i}^{(t)} \cdot g_{r_{\tau+1}, i, r_{\tau+2}, c}^{(\tau+1)} \cdot h_{r_0, b, r_{\tau+1}}^{(\tau)} \quad (\text{Ec. 4.48})$$

De la expresión anterior (Ec. 4.48) se puede deducir que el orden de esta operación en particular es de $\mathcal{O}(R'^2 \cdot R \cdot m \cdot C \cdot B)$. El último paso que es la operación de traza, donde cada uno de los elementos del último estado oculto $\mathbf{H}^{(\tau+2)} \in \mathbb{R}^{B \times C}$ queda definido como:

$$\begin{aligned} h_{b,c}^{(\tau+2)} &= \sum_{r_{\tau+2}=1}^{R_{\tau+2}} h_{r_{\tau+2}, b, r_{\tau+2}, c}^{(\tau+1)} \\ &= \text{Tr}(\mathbf{H}^{(\tau+1)}(b, c)) \end{aligned} \quad (\text{Ec. 4.49})$$

El orden de esta operación (Ec. 4.49) es de $\mathcal{O}(R' \cdot C \cdot B)$, y éste, al igual que al momento de calcular el orden anterior (Ec. 4.41), es despreciable. Teniendo en cuenta el resto de los ordenes calculados, el orden total para la arquitectura compartida de la descomposición de *Tensor Ring* es:

$$\mathcal{O}(((R' \cdot R^2 + n) \cdot \tau + R'^2 \cdot R \cdot C) \cdot m \cdot B) \quad (\text{Ec. 4.50})$$

A la hora de comparar el orden de ambas arquitecturas la *Tensor Train compartida* (Ec. 4.47) y la *Tensor Ring compartida* (Ec. 4.50), es importante notar la diferencia en el orden de operaciones entre ambos. El segundo tiene un orden casi cúbico con respecto al rango de los tensores, mientras que el primero tiene un orden cuadrado con respecto al rango, esta diferencia se puede pensar como una consecuencia directa de la operación extra que representa la traza en el caso de la descomposición TR.

A continuación se puede realizar de la misma manera que se comparan el orden de las operaciones, el mismo análisis para el tamaño mismo de la red. La definición del tamaño de la red es la cantidad de variables que son modificadas a la hora del proceso de entrenamiento y aprendizaje, no se toman en cuenta los estados o entradas que son conformadas con la técnica denominada de *padding*, ya que éstas son constantes y no son parámetros aprendibles, de la misma manera las funciones no lineales o de *softmax* se aplican sobre las mismas salidas, no involucran parámetros en sus operaciones. Para empezar ambas redes, tienen como primer paso en el procesamiento de sus parámetros la misma operación, la aplicación del *Feature Map* (Ec. 4.23) sobre la entrada de la red, la cantidad de parámetros se puede calcular como:

$$\text{Parámetros}(\text{Feature Map}) = m \cdot n + m \quad (\text{Ec. 4.51})$$

Es notable que el valor de m es un hiperparámetro importante de la red, ya que define la cantidad de *features* que tiene cada división de la entrada. Mientras que el n es otro hiperparámetro de la red pero que se calcula implícitamente de otros dos, la cantidad de divisiones por la cual se fracciona cada entrada y el tamaño de la misma que depende de la base de datos utilizada. Se deduce de esto que el tamaño de la red va a depender directamente de la base de datos utilizada.

Estudiando la Arquitectura *Tensor Ring compartida* de la Figura 27 y comparándola con la Arquitectura de *Tensor Train*, en la Figura 32, podemos notar que ambas comparten el tensor núcleo utilizado como el componente recurrente de la red. Este núcleo en ambos casos tienen la misma cantidad de parámetro, se pueden calcular de la siguiente manera:

$$\text{Parámetros(Tensor compartido)} = R^2 \cdot m \quad (\text{Ec. 4.52})$$

La diferencia entre el último tensor de ambas arquitecturas no sólo afecta como se desarrolló en la sección anterior al orden de operaciones, también es el principal diferencial a la hora de pensar en el tamaño entre las dos. En el caso de *Tensor Ring*, el último tensor en comparación con el caso de *Tensor Train* tiene el rango que le permite realizar la operación de traza en el último paso, esto le agrega una dimensión más, aumentando considerablemente el tamaño de la red:

$$\text{Parámetros(último Tensor TR)} = R' \cdot R \cdot m \cdot c \quad (\text{Ec. 4.53})$$

$$\text{Parámetros(último Tensor TT)} = R \cdot m \cdot c \quad (\text{Ec. 4.54})$$

Considerando el primer tensor (Ec. 4.45) de la red *Tensor Train*, y las ecuaciones previamente desarrolladas se puede calcular el tamaño de ambas redes:

$$\text{Parámetros(TR)} = m \cdot (R^2 + R \cdot c \cdot R' + (n + 1)) \quad (\text{Ec. 4.55})$$

$$\text{Parámetros(TT)} = m \cdot (R^2 + R \cdot (c + 1) + (n + 1)) \quad (\text{Ec. 4.56})$$

Como es el caso para el orden de las operaciones (Ec. 4.50) (Ec. 4.47), la cantidad de parámetros es también mayor en un grado (en el término del último tensor) para la arquitectura de *Tensor Ring* que de *Tensor Train*, estas dos diferencias se pueden ver observadas en la sección de experimentos de este trabajo.

CAPÍTULO 5: RESULTADOS EXPERIMENTALES

En este capítulo se verifican experimentalmente los resultados teóricos expuestos hasta ahora validando las arquitecturas basadas en TT y TR sobre bases de datos conocidas. En la primera sección se describen en detalle las bases de datos utilizadas en este trabajo. Luego se especifican los detalles de implementación de las redes con las cuales se realizaron los experimentos y sus parámetros de evaluación. Finalmente se exponen los resultados de los ensayos realizados, incluyendo conclusiones particulares para cada una de las bases de datos utilizadas.

5.1. Bases de datos utilizadas

Las redes mencionadas en el trabajo se desarrollaron utilizando **PyTorch** [78], un paquete de *Python* que provee computación tensorial con aceleración de *GPU* y redes neuronales profundas con algoritmos de aprendizaje incorporados. El trabajo realizado en este sentido consistió principalmente en adaptar las arquitecturas antes mencionadas con el fin de comparar su funcionamiento utilizando varios tipos de bases de datos. El código se encuentra disponible en este repositorio: github.com/AndresOtero/TensorDecompositionMachineLearning.

5.1.1. MNIST y Fashion-MNIST

Las bases de datos utilizadas en el trabajo son estándares a la hora de entrenar y poder probar el funcionamiento de diferentes arquitecturas de redes neuronales. La primera base de datos sobre el cual se realizaron pruebas es *MNIST* [66] (Figura 33), que es una base de datos de dígitos escritos a mano, estandarizadas en tamaño y centrados en una escala de grises. Es una mezcla de dígitos sacadas de 2 bases diferentes del *National Institute of Standards and Technology* de dígitos escritos por alumnos de secundaria y otra de empleados del *United States Census Bureau*. Éste es utilizado para probar todo tipo de algoritmos de aprendizaje y de reconocimiento de patrones, cuenta con un conjunto de ejemplos de 60.000 imágenes y un conjunto de test de 10.000 imágenes. Esta base consta de 10 clases, una por cada dígito, y es el más citado en trabajos científicos. El tamaño pequeño y su disponibilidad en las diferentes librerías y paquetes lo vuelve muy atractivo para realizar pruebas rápidas sobre todo tipo de arquitecturas.

Esta base de datos es utilizada para probar diferentes técnicas de calificación pero es recomendado no usarlo como única medición de la efectividad de un algoritmo, debido a que se suele obtener buenos resultados con algoritmos no muy sofisticados. Por ejemplo, un clasificador lineal puede llegar a lograr un 88 % de precisión [65, 66]. Empleando redes más complejas se han logrado hasta un 99.7 % de precisión [17, 94]. El consenso general es que, si una red no funciona para esta base, probablemente no funcione en general y, si funciona, todavía puede llegar a fallar en otros.

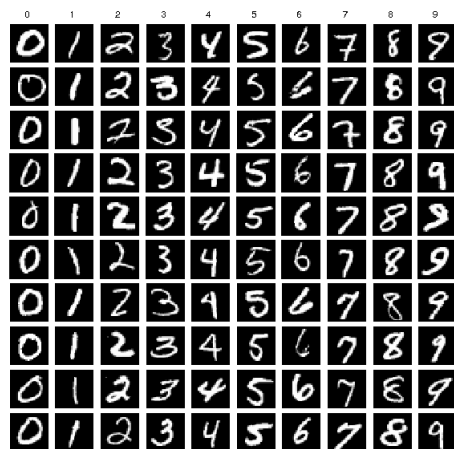


Figura 33: Ejemplo de un grupo de imágenes agrupadas por dígitos en *MNIST* [66].

Debido a todos estos factores se decidió agregar *Fashion-MNIST* [95] (Figura 34), es una base de datos que tiene el mismo formato que *MNIST*, tiene 10 clases, un conjunto de ejemplos de 60.000 imágenes, un conjunto de test de 10.00 imágenes estandarizados en tamaño y centrados en una escala de grises. En este caso las imágenes son fotos de 10 categorías de diferentes prendas de vestir de una

tienda de moda: remera, pantalón, suéter, vestido, abrigo, sandalias, camisa, zapatillas, mochila y botas bajas. Su propósito es funcionar como un reemplazo transparente al original. Debido a que tienen exactamente el mismo formato, se puede simplemente poner en lugar del *MNIST* y sin ningún cambio a la red o algoritmo probar su funcionamiento.



Figura 34: Ejemplo de un grupo de imágenes de *Fashion-MNIST*[95].

Se realizaron pruebas al respecto comparando el funcionamiento de diferentes algoritmos utilizando ambas bases de datos [95], donde una regresión logística obtuvo 91.7% con el conjunto de test de *MNIST*, mientras que sobre el conjunto de test de *Fashion-MNIST* obtuvo 84,2% de precisión. También en el mismo trabajo se realizaron pruebas sobre una red MLP (ver sección 2.1), donde se obtuvo 97.2% de precisión en el conjunto de test de *MNIST*, mientras que sobre el conjunto de test de *Fashion-MNIST* obtuvo 87,1% de precisión. En ambos ejemplos se puede ver como la base de datos introducida es más difícil de clasificar que la original, proveyendo de esta manera un caso más complejo sin tener que obligatoriamente modificar la arquitectura o el preprocesamiento de la red o algoritmo que estaba siendo utilizado. Por esta razón se decidió incluirlo en las pruebas realizadas sobre las arquitecturas implementadas.

Todas las bases de datos utilizadas para probar las redes implementadas proveen como entrada, de una u otra manera, una foto con escala de grises, una foto a color o un conjunto de palabras, un tensor de diferentes dimensiones que debe ser dividido en una cierta cantidad de pasos τ para luego ser transformado por el *feature map* e ingresado como entrada a alguna de las arquitecturas implementadas. Para el caso de una imagen como en el caso de *MNIST*, es necesario hacer una modificación para poder transformar desde su formato original de una matriz de 28×28 a un formato que este dividido en τ divisiones. Esta transformación es similar a la que se realiza en el trabajo de Khrulkov *et al.* 2017 [52], el primer paso es dividir la imagen en parches, en una cantidad r de divisiones por fila y c de divisiones por columna, teniendo en cuenta que $r \cdot c = \tau$. Cada uno de estos parches se aplanan y luego se concatena formando una matriz de $\tau \times n$ siendo $n = \frac{28 \cdot 28}{\tau}$ en el caso de *MNIST*, éste cambiará dependiendo del tamaño de la matriz original. Un ejemplo con $r = 4$, $c = 4$, $\tau = 16$, $n = 49$ para un muestra de un número 0 para *MNIST* se puede observar en la

siguiente figura:

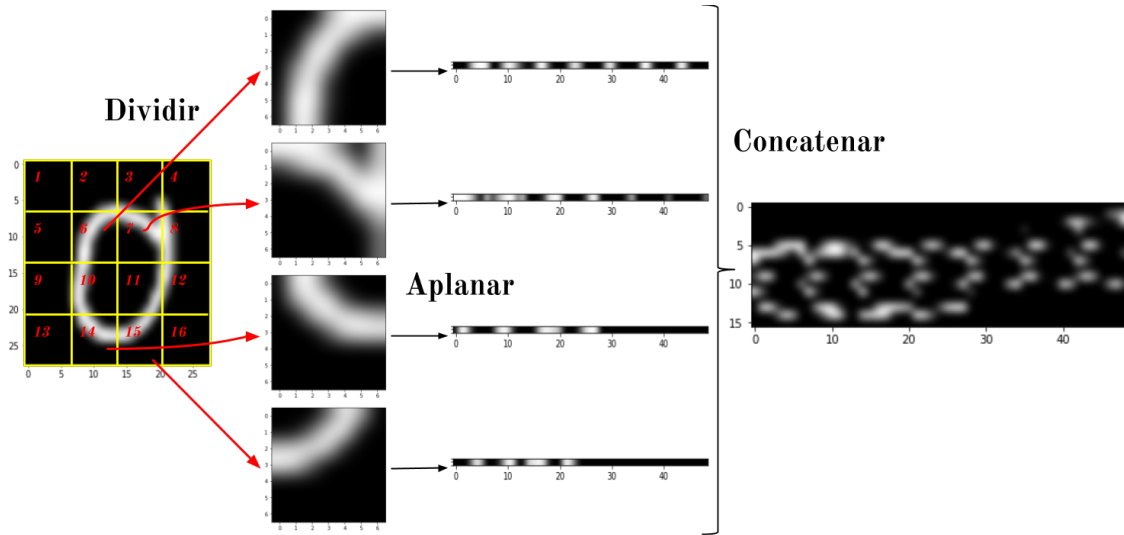


Figura 35: Ejemplo de aplanamiento de divisiones de una muestra de *MNIST*, la imagen está difuminada para que sea más simple de interpretar.

Es interesante notar que la transformación ilustrada (Figura 35) es equivalente a utilizar una capa convolucional sin solapamiento. El hecho de aplanar los parches cambia la geometría de la imagen, un humano a simple vista pierde la capacidad de reconocer un número en el caso de la base de datos de *MNIST*. Pero si se realiza una observación más profunda de diferentes muestras de varios números (Ver Figura 36), se puede concluir que hay patrones reconocibles en los diferentes ejemplos exhibidos. A pesar de tener diferentes casos de cada una de las clases siendo distintas ellas entre sí, aún se pueden identificar como se menciono anteriormente, patrones específicos en alguno de los pasos que caracterizan a las imágenes de la misma clase. Lo mismo pasa cuando se evalúan diferentes ejemplos provenientes de la base de datos de *Fashion-MNIST* (Ver Figura 37), si se ven el patrón que genera una remera (Izq.) es diferente al que forma un pantalón (Der.).

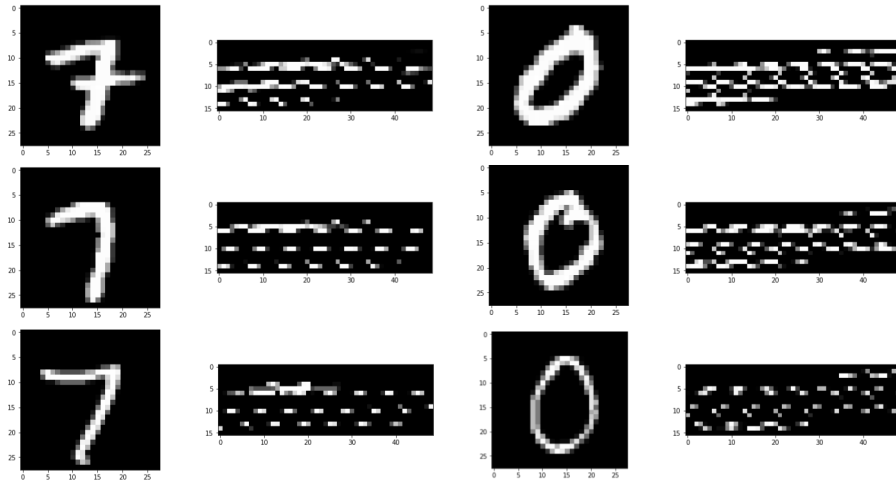


Figura 36: Ejemplo varias muestras de *MNIST* aplanadas. Del lado derecho se pueden ver varios ejemplos de un número 7 y del lado izquierdo se pueden ver ejemplos de un número 0.

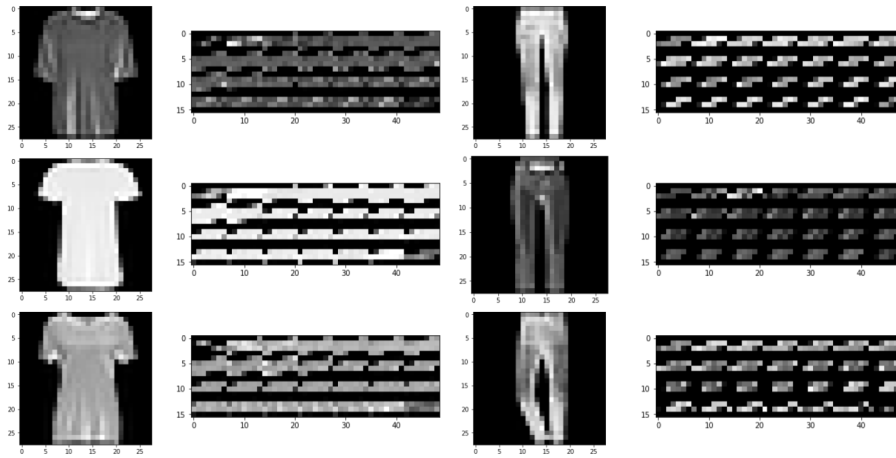


Figura 37: Ejemplo varias muestras de *Fashion-MNIST* aplanadas. Del lado derecho se pueden ver varios ejemplos de una remera y del lado izquierdo se pueden ver ejemplos de un pantalón.

Algo importante a notar es que en las figuras 36 y 37 la imagen no está normalizada, la convergencia del algoritmo de *backpropagation* es mejor si el promedio de cada entrada, sobre todo el set, es cercano a cero [67]. En el caso de *MNIST*, por ejemplo, se utiliza como media ($\mu = 0,1307$) y desvío estándar ($\sigma = 0,3081$), que son los valores para toda la base de datos, para cada píxel x_{ij} de una matriz ingresada $\mathbf{X} \in \mathbb{R}^{I \times J}$ se recalcula como:

$$x_{ij} = \frac{x_{ij} - \mu}{\sigma} \quad (\text{Ec. 5.1})$$

En el caso de *Fashion-MNIST*, también se calculó la media ($\mu = 0,2860$) y desvío estándar ($\sigma = 0,3205$) para todo el conjunto. Esta modificación hace que la media de los datos sea 0 y el desvío estándar sea 1, logrando el objetivo que el promedio de cada entrada sobre todo el conjunto sea cercano a cero.

5.1.2. CIFAR-10

CIFAR-10 [59] es una base de datos que consiste de 60.000 imágenes a color divididas en 10 clases (ver Figura 38), al igual que los mencionados anteriormente tienen 50.000 imágenes en un conjunto de entrenamiento y 10.000 en un conjunto de test. Las clases son una mezcla de animales: pájaro, gato, ciervo, perro, rana y caballo; y medios de transporte: avión, auto, bote y camión. Cada clase tiene 5.000 imágenes para el conjunto de entrenamiento y 1.000 imágenes para el conjunto de test. La diferencia de esta base con las anteriores que tenían el mismo formato que el *MNIST*, es que las imágenes de 32x32 están coloreadas en formato RGB. Esto trae un nuevo desafío a la hora de probar con las arquitecturas implementadas, porque cambia la geometría del problema y la dificultad de los objetos a reconocer.

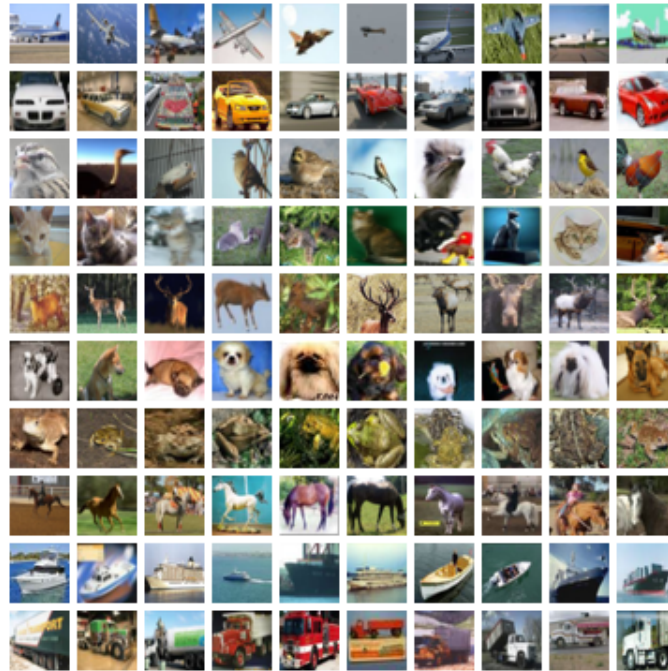


Figura 38: Ejemplo de un grupo de imágenes de *CIFAR-10*[59].

Base de datos	Precisión	Red	Cantidad de parámetros
<i>MNIST</i>	99.7 %	MCDNN [17]	171,940
<i>Fashion-MNIST</i>	96.35 %	WRN-28-10 [99]	36.5M
<i>CIFAR-10</i>	99.0 %	AmoebaNet-B(18, 512) [46]	557M
<i>IMDB</i>	97.4 %	NB-SVM [91]	-

Tabla 5: Estado del arte de las bases de datos utilizadas

En la implementación a la hora de clasificar la base de datos de *CIFAR-10* [59] se identificó en

los experimentos iniciales un fenómeno que en modelos de *deep learning* se denomina *overfitting* o *sobreajuste*. En las primeras corridas realizadas se registró una importante diferencia en la precisión de la clasificación en el conjunto de entrenamiento con los de prueba, la eficacia del modelo para el primero era considerablemente superior al del segundo, el modelo estaba sobre ajustando sus parámetros sobre los casos de entrenamiento. Esto implica que el modelo no es capaz de generalizar correctamente para todo el dominio de casos lo que extrajo del conjunto de entrenamiento.

La mejor manera para que categorice mejor es poder entrenarlo con muchos más datos, pero la cantidad de datos es limitada. La solución a este problema es crear más datos para agrandar el conjunto de entrenamiento. El problema con la base *CIFAR-10* es que crear desde cero nuevas imágenes que entren dentro de las categorías existentes es una tarea compleja. Para un modelo como el de este trabajo, en el cual un clasificador debe tomar un tensor multidimensional (imagen) y asignarle una categoría, esto significa que éste debe permanecer sin varianza ante una gran variedad de transformaciones.

Es posible crear nuevos datos transformando de diferentes maneras la entrada y no modificar la categoría de la misma, esta técnica es conocida como *data augmentation* [32]. Este método es particularmente efectivo para problemas de reconocimiento de imágenes, como es el caso en *CIFAR-10*. Esto se debe a que las imágenes son tensores multidimensionales que tienen múltiples factores de variación, los cuales pueden ser fácilmente simulados. Operaciones como trasladar las imágenes de entrenamiento unos pocos píxeles en cada dirección pueden a menudo mejorar en gran medida la generalización, incluso si el modelo ya ha sido diseñado para ser parcialmente invariante de la traslación utilizando las técnicas de convolución y *pooling*. Muchas otras operaciones, como rotar o escalar la imagen también han demostrado ser bastante efectivas. Esta técnica no es útil para casos como el de reconocimiento de caracteres o jeroglíficos, como es el caso de *MNIST*, la diferencia entre un '6' y un '9' se torna imposible si se gira la imagen 180 grados, así que las operaciones para incrementar el tamaño de los datos de entrenamiento deben implementarse teniendo en cuenta el tipo de base de datos usado. Un detalle importante a ser notado es que las transformaciones previamente descritas, son solamente utilizadas a la hora de entrenar y validar a la red pero no a la hora de testearla, la idea detrás de esta técnica es crear en base a los datos para entrenar un conjunto más amplio de entradas, ésto claramente no es necesario a la hora de testear el funcionamiento de la red y tiene muchas chances de ser contraproducente.

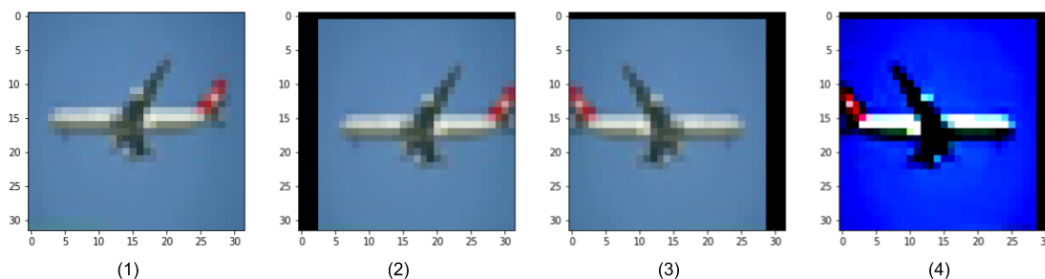


Figura 39: Ejemplo de una muestra de *CIFAR-10* normalizada y aumentada. (1) Imagen de un avión parte de la base de datos. (2) Imagen recortada de manera aleatoria. (3) Imagen aleatoriamente espejada horizontalmente. (4) Imagen normalizada.

El desafío que plantea la base *CIFAR-10*, a diferencia de *MNIST* o *Fashion-MNIST* es que sus imágenes tienen una dimensión más que el de los otros conjuntos, es una imagen a color como bien se puede apreciar en la Figura 39. En contraste con los otros, que tienen una dimensión que representa una escala de grises, el *CIFAR-10* tiene tres canales, cada uno representando un color: azul, rojo o verde. Esto provee una complejidad dimensional a la hora de procesar la entrada de la red que

no fue encontrada en los ejemplos anteriores, por lo tanto el esquema anteriormente detallado no funciona naturalmente y requiere de ciertas adaptaciones.

5.1.3. IMDB

Todas las bases de datos mencionadas anteriormente están compuestas de imágenes pertenecientes a diferentes clases, que agregan diferentes desafíos para probar las arquitecturas implementadas. La base de datos de *Large Movie Review Dataset* [70] es un conjunto de 50.000 críticas publicadas en el sitio *IMDB*, con menos de 30 críticas por película. Tiene una cantidad igual de críticas positivas y negativas. Una crítica negativa tiene un puntaje menor o igual a 4 de 10 y una crítica positiva tiene un puntaje mayor o igual a 7 de 10, no hay críticas que pueden ser consideradas neutras dentro de la base. En el paper original que introdujo esta base [70] se utiliza un algoritmo que utiliza una mezcla de técnicas no supervisadas y supervisadas, para poder capturar información semántica, que puede ser utilizada en el análisis de sentimientos de un “documento” (una crítica de una película en este caso). Este algoritmo es utilizado con distintos *features*, logrando con diferentes variantes hasta un 88 % de precisión en la clasificación de sentimientos. En algunos trabajos más nuevos, se mejoró esta marca logrando hasta un 97.4 % de precisión en la clasificación, utilizando un modelo de *Naive Bayes-Support Vector Machine* (NB-SVM), un clasificador SVM alimentado por bolsas de n-gramas ponderadas por *Naive Bayes* [91]. Este tipo de problema, la clasificación de sentimientos de un texto, está dentro del conjunto de problemas que aborda el campo del procesamiento de lenguaje natural o *Natural Language Processing* (NLP), con la intención de abordar problemáticas en este campo se agregó este dataset entre los experimentos realizados.

El tratamiento y el preprocesamiento para las bases de datos para *NLP* es muy diferente al de otros para las bases de datos de imágenes. El desafío es cómo hacer para que un texto o una imagen se conviertan a un formato que pueda ser utilizado como entrada de una red neuronal y qué tipo de tratamiento debe realizarse para permitir a la red entrenarse para lograr los mejores resultados posibles.

Lo primero en el procesamiento del texto es realizar lo que llama *tokenization*, esto es básicamente segmentar el texto de la crítica en diferentes *tokens*. Los *tokens* pueden ser palabras, signos de puntuación, dígitos, entre otros y también pueden ser estos token clasificados como adjetivos, nombres propios, verbos, símbolos, etc. Esto puede para un humano sonar trivial pero para una computadora es una tarea compleja que permite empezar a procesar el texto para que pueda ser utilizada por ella. Este proceso depende del lenguaje que en el caso de este texto es inglés, el software utilizado para esta tarea es llamado *Spacy* [45], un ejemplo de este proceso se puede ver en la Figura 40. Se eligió *Spacy* por ser una librería de código abierto para *NLP* en *Python* que es bastante popular y se usa en ambientes de producción que necesitan este tipo de procesamiento.

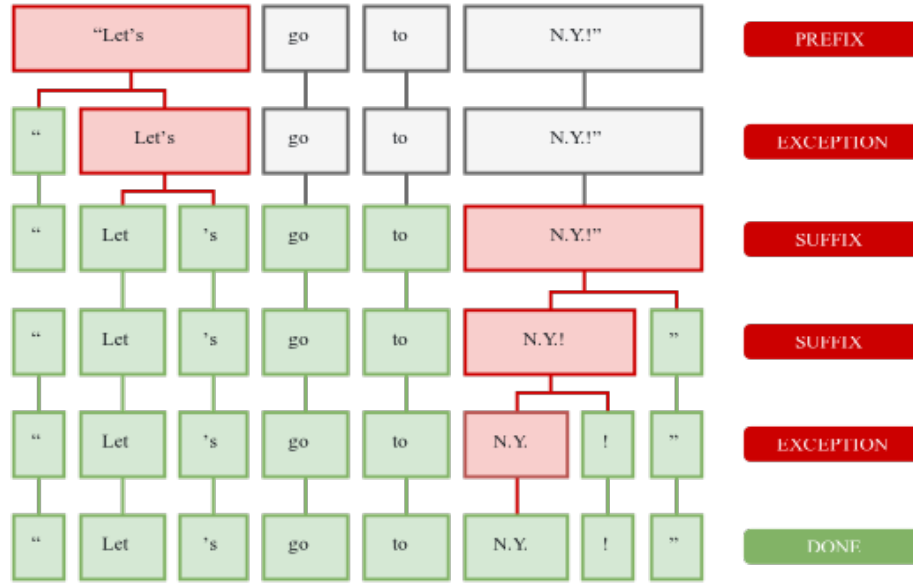


Figura 40: Ejemplo de como *Spacy* [45] realiza la *tokenization* de una oración de ejemplo en inglés y como resuelve ciertos conflictos complejos y anidados.

Una vez que se convierte el texto en *tokens* falta un paso más para terminar la conversión a un tensor que pueda funcionar como entrada de una red neuronal. Para la tarea de mapear los *tokens* en vectores de números reales se utilizó *word embeddings*, conceptualmente se realiza un encaje matemático desde un espacio con múltiples dimensiones por palabra, a uno de mucho menor dimensión. Para este fin se usó *GloVe: Global Vectors for Word Representation* [79], el cual es un algoritmo de aprendizaje no supervisado para transformar palabras en vectores representativos. El entrenamiento se realiza en base a estadísticas globales de coincidencia global palabra-palabra de un corpus, los vectores resultantes muestran subestructuras lineales en el espacio vectorial de palabras muy interesantes.

Esta herramienta resuelve el problema de reducir la dimensión de los vectores pasando de tener un vector del tamaño del vocabulario, denominado *one-hot vector*. En el caso de utilizar esta metodología en un vocabulario como el de *GloVe.6B* implicaría un vector de 400.000 dimensiones, en lugar de uno de 50 dimensiones que fue el utilizado en este trabajo. Sea $l \in \mathbb{R}$ la longitud del texto, $|V| \in \mathbb{R}$ el tamaño del vocabulario y $l \in \mathbb{R}^{k \times |V|}$ la matriz de *embedding* de un vector de palabras de k dimensiones. La i -ésima palabra del texto es transformada en un vector de k dimensiones w_i :

$$w_i = W \times x_i \quad (\text{Ec. 5.2})$$

La reducción de dimensiones no es el único beneficio o característica de esta herramienta. La transformación de un token a un vector permite extraer relaciones entre palabras en base a la relación entre sus representaciones vectoriales en este nuevo subespacio. Una de las más potentes, y al mismo tiempo simple, es que la distancia euclidiana de dos vectores de palabras provee una manera de medir la similitud tanto lingüística, como semántica entre ellos. El ejemplo que presenta en la documentación es que la palabras más cercanas a *frog* (rana) son *frogs* (ranas), *toad* (sapo) y *litoria*, que es una especie de ranas, esto muestra que una clara relación semántica entre las palabras

es plasmada en una relación matemática entre los vectores.

La distancia euclidiana (o similitud de coseno) es una de las maneras simples entre las cuales se observar la relación entre las palabras, pero hay relaciones más complejas que pueden ser detectadas a través de la relación entre los vectores de cada palabra. El ejemplo utilizado en el trabajo referenciado es el de la relación entre *man* (hombre) y *woman* (mujer), se pueden considerar términos similares en cuanto a que ambos son usados para clasificar humanos, pero pueden también ser considerados opuestos ya que son términos en muchos casos utilizados de manera antónima. La manera en la cual se puede capturar la relación aplicando esta herramienta en vez de observar la distancia entre ambos vectores es utilizando la diferencia vectorial entre ambos. La idea detrás de esto es que el concepto fundamental que diferencia *man* (hombre) y *woman* (mujer), se replica en otras relaciones como *king* (rey) y *queen* (reina), o *brother* (hermano) y *sister* (hermana). No significa que los conceptos sean iguales, pero que las diferencias entre ellos son muy similares, lo cual se puede ver en la Figura 41. Este tipo de relación geométrica, donde la diferencia entre vectores de un conjunto de conceptos se puede observar en otros ejemplos, como el de código postal y su ciudad correspondiente, o la relación entre un adjetivo, su comparativo y su superlativo. Otra característica de los vectores resultantes de esta palabra es que esta diferencia permite que exista una aritmética de palabras donde el vector reina es muy cercano al resultante de hacer la operación entre vectores rey-hombre+mujer=reina.

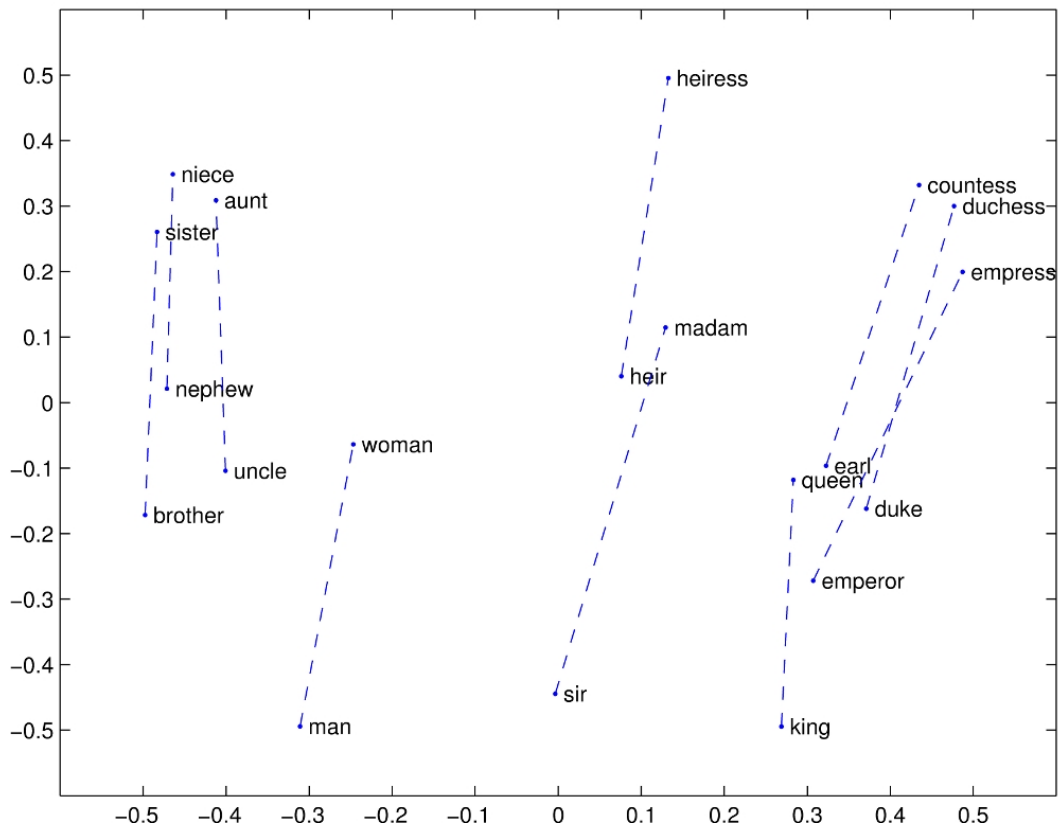


Figura 41: Ejemplo de como *GloVe* [79] muestra los mismos patrones de diferencia con varios conceptos con la misma relación.

Un ejemplo de una crítica negativa tokenizada es el siguiente:

['Easily', 'the', 'worst', 'movie', 'I', 'have', 'ever', 'seen', 'in', 'my', 'life', '.', 'Direction', ':',

'none', '.', 'Story', ':', 'pathetic', '.', 'Screenplay', ':', 'that', 'will', 'be', 'a', 'good', 'idea',
 ', ', 'There', 'is', 'a', 'lot', 'of', 'gratuitous', 'graphics', ', ', 'all', 'of', 'pathetic', 'quality', ', ',
 'Preserve', 'your', 'sanity', ', ', 'do', 'nt', 'ever', 'see', 'this', 'movie', '!','],

Analizando la base de datos surge un problema que se vuelve visible evaluando la Figura 42. La distribución de la longitud de las críticas tiene mucha varianza, la mediana es 201 y el desvío estándar es de alrededor de 199. Para poder lidiar con este desafío y poder poner como una entrada de tamaño uniforme para la red, se truncan las longitudes de las críticas a 200, si una de éstas es mayor a 200 se quedan con los primeros 200 tokens mientras que si éste resulta menor se lo rellena hasta que tengan esta longitud. El fundamento detrás de este preprocesamiento está basado en la naturaleza de la base de datos, es difícil pensar que no se puede determinar el sentimiento de una crítica luego de leer alrededor de las primeras doscientas palabras, siendo que más del 90 % de las críticas no supera los 600 caracteres. Basados en toda esta información se decidió para todos los experimentos con este set realizar este tratamiento de los datos, considerando como aceptable esta pérdida de información.

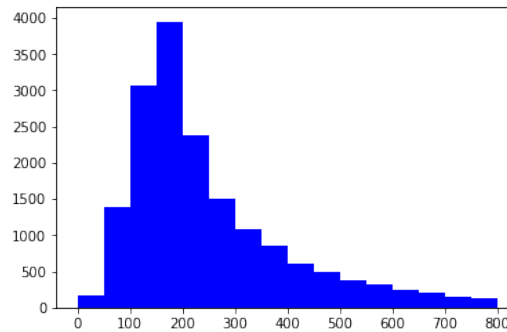


Figura 42: Histograma de cantidad de críticas respecto a la longitud de la misma.

Para realizar los experimentos con esta base, se utiliza un 70 % del conjunto de entrenamiento de las 25.000 críticas para esta tarea y el 30 % restante se utiliza para validar. En el proceso de convertir los tokens en vectores que puedan ser procesados por la red se debe establecer el tamaño del vocabulario. En el set hay más de 160.000 tokens, pero una vez más analizando los datos se puede identificar que la mayoría de las ocurrencias están concentradas en un número limitado de palabras. En el set hay más de 1.000.000 de ocurrencias sumando todos los tokens de cada uno de los textos, para limitar el vocabulario se eligen aquellos 25.000 con mayor cantidad de apariciones en las críticas seleccionadas para el conjunto de entrenamiento, éstos representan casi el 97 % de las ocurrencias en el total de la base. Esta importante reducción del vocabulario mejora el funcionamiento y el rendimiento de la capa de *embedding* de la red. Además de los tokens antes mencionados se le agregan dos más para llevar el total de valores únicos a 25.002, uno de éstos es un token para representar a todos los valores excluidos, que son los que menos frecuencia tienen en el set, éste tiene un valor que se denomina *único*. El otro valor agregado al vocabulario es uno denominado *pad* o relleno, éste se usa para rellenar los ejemplos del set que tienen menos de 200 palabras, el tamaño fijo elegido para todos los ejemplos del set, se utiliza este valor para completar el vector.

Una vez definido el vocabulario del set entra en escena la capa de *embeddings*, inicializada con una serie de vectores previamente entrenados a los que se hizo referencia anteriormente. Estos vectores son los correspondientes a los tokens elegidos en el vocabulario, por eso es tan importante elegir una cantidad limitada de éstos que se consideren relevantes, elegir muy poco puede reducir la dimensionalidad y la cantidad de información a la red mientras que un vocabulario muy extenso

puede llevar a problemas de funcionamiento y en ciertos casos a que la red pueda hacer *overfitting*. Los tokens de relleno y el único están incluidos en el vocabulario lo que provoca que la capa termine siendo una matriz de 25002 filas, la cantidad de tokens, y 50 columnas, la dimensión de los vectores entrenados elegida de *GloVe.6B*. Un detalle importante a notar respecto de estos 2 tokens singulares es que éstos no son parte del vocabulario entrenado utilizado en los *word embeddings*, como se espera que éstos no influyan en determinar el sentimiento de una crítica se los inicializa como un vector de ceros.

En el momento de utilizar los *word embeddings* con la arquitectura propuesta se puede apreciar que este tipo de datos es más apto para una red recurrente, se pueden fácilmente separar la entrada en diferentes grupos fácilmente divisibles en un formato secuencial. Pero surge una complicación que ya fue mencionada en este trabajo, el *vanishing gradient problem* de las redes recurrentes menos complejas como la propuesta, a diferencia de una como LSTM, se vuelve muy notorio en secuencias muy grandes, lo que ocasiona que no sea posible dividir de una manera que se puede considerar como natural para esta base de datos, el de ingresar cada palabra como una entrada de la red. Para adaptar este problema a las limitaciones inherentes de la red, se agrupan las palabras en conjuntos más grandes. Esta agrupación secuencial en grupos de palabras se puede interpretar como “oraciones”, no se realiza un análisis sintáctico de las críticas recibidas, pero la irregularidad de la base de datos en cuanto al formato y el largo de las críticas hace que no sea fácil encontrar una partición regular que tenga sentido. Aún así esta estrategia genera que se pierda información intrínseca en la red, los resultados, como se detallan más adelante, no fueron del todo satisfactorios ni los esperados para el trabajo.

5.2. Arquitecturas de referencia

Para poder comparar las arquitecturas propuestas, cotejando su capacidad de clasificar con otro tipos de redes, que son utilizadas tanto en la academia como profesionalmente y poder tener un punto de referencia que no sean el estado del arte, ya previamente aclarado en la Tabla 5.

5.2.1. MNIST y Fashion-MNIST

Se experimentó con una red RNN para poder comparar su efectividad para las bases *MNIST* y *Fashion-MNIST*, teniendo ésta la forma denominada anteriormente como *Vanilla RNN*, detallada en la Sección 2.4, utilizando los *Features Maps* como entrada para la red. Esto permite hacer un paralelo de cómo funcionan los modelos planteados contra esta red que puede ser considerada como “genérico” en este contexto de uso. Sus parámetros son comparables, siendo m el tamaño del vector de entrada \mathbf{x}^{t-1} y el rango R el tamaño del estado oculto \mathbf{h}^{t-1} . También se precisa la cantidad de pasos τ que son la cantidad de divisiones en la imagen original.

Se ensayaron también experimentos con una red LSTM, descrito en la sección A.2. Comparte de la misma manera que la red RNN los hiperparámetros que con las arquitecturas planteadas, siendo m el tamaño del vector de entrada \mathbf{x}^t , el rango R el tamaño del estado oculto \mathbf{h}^{t-1} y la cantidad de pasos τ . Para ambos RNN y LSTM se probaron utilizar diferentes tamaños de M , para evaluar su funcionamiento con diferente cantidad de *features*.

5.3. CIFAR 10

El primer intento fue usar el mismo diseño utilizado para las bases *MNIST* y *Fashion-MNIST*, incluido el preprocesamiento con el aplanamiento de divisiones, pero redimensionando el tensor de entrada. La imagen de la base de datos tiene 3 dimensiones, de forma $\mathcal{X} \in \mathbb{R}^{3 \times 32 \times 32}$, éste es transformado a una matriz $\mathbf{X} \in \mathbb{R}^{3 \times 1024}$. Para la misma red, los resultados experimentales no fueron

buenos, su rendimiento estaba alrededor del 50 % de precisión, pudiendo éste definirse como que la red está funcionando de manera correcta y supera la barrera mínima para suponer que cumple con su objetivo, pero no son satisfactorios para los objetivos planteados para este trabajo.

Con el objetivo de poder mostrar mejores resultados, además de experimentar con otros tipos de arquitecturas, integrando la arquitectura completa en modelos más complejos, comparándolo con diferentes redes, testeando su capacidad y desempeño en estos contextos, se utilizó como referencia una red CNN que podría considerarse como genérica, realizando a continuación un diagrama del modelo utilizado:

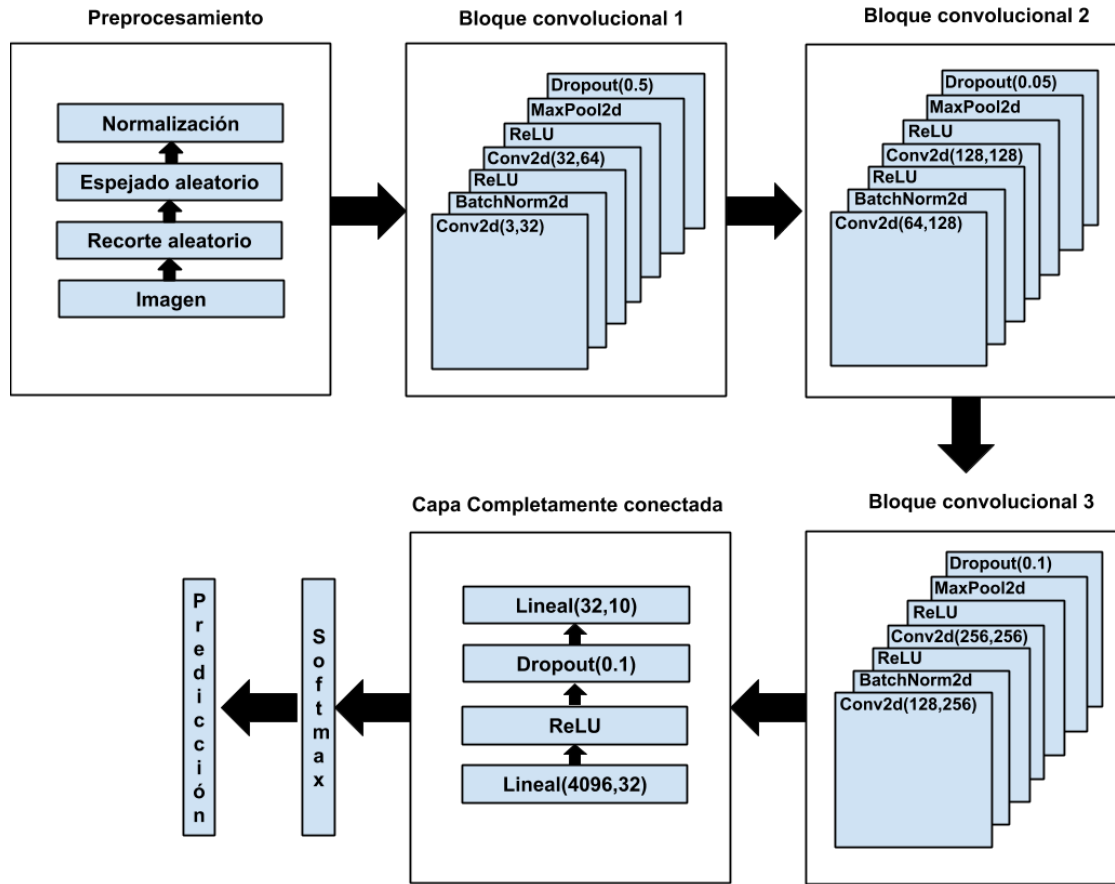


Figura 43: Diagrama de la CNN de referencia.

El preprocesamiento mostrado en la Figura 43 es el que se describió en el ejemplo que se encuentra en la Figura 39. La CNN puede describirse en 4 bloques lógicos por los cuales se procesa la entrada normalizada y se hace una predicción luego de utilizar una función *softmax* logarítmica (Ec. A.3). Las tres capas convolucionales son idénticas en sus componentes y su orden, pero con variaciones en ciertos parámetros. El componente denominado *Conv2d* es el que realiza la tarea de aplicar la convolución sobre el tensor de entrada, esta operación es parecida a la descrita en (Ec. 2.8), pero con un par de variaciones, debido a la diferencia de dimensiones de la imagen. Ésta realiza un cambio en el tamaño de la dimensión, en la primera convolución de la primera capa. Por ejemplo, la entrada son los 3 colores de la imagen y la salida es una imagen de grosor 32. Para describir genéricamente la

operación, se define la entrada como $\mathcal{I} \in \mathbb{R}^{C_{in} \times H_{in} \times W_{in}}$, en el primer caso C_{in} son los 3 canales de entrada, uno por cada uno de los colores, mientras que $H_{in} \times W_{in}$ es el tamaño de la imagen, en el caso de *CIFAR* es 32×32 . El tensor de pesos de la convolución tiene la forma $\mathcal{W} \in \mathbb{R}^{C_{out} \times C_{in} \times K_0 \times K_1}$, donde el *kernel* por el cual se realiza la convolución es de tamaño $K_0 \times K_1$ que, para todos los casos en esta red, es de 3×3 . En el mismo estilo que con una capa completamente conectada, se puede definir un vector de *bias* $\mathbf{b} \in \mathbb{R}^{C_{out}}$, donde la salida es entonces $\mathcal{O} \in \mathbb{R}^{C_{out} \times H_{out} \times W_{out}}$. La cantidad de canales de salida C_{out} en el primer caso es de 32, mientras que el tamaño de la imagen de salida es de $H_{out} \times W_{out}$, éstas dependen del tamaño del núcleo de la convolución, y se calculan de la siguiente manera:

$$\begin{aligned} H_{out} &= H_{in} - K_0 + 1 \\ W_{out} &= W_{in} - K_1 + 1 \end{aligned} \quad (\text{Ec. 5.3})$$

Teniendo en cuenta todas estas definiciones, en base a la operación definida en la ecuación (Ec. 2.8), la operación de convolución *Conv2d* se calcula como:

$$\mathcal{O} = \mathbf{b} + \sum_{k=0}^{C_{in}-1} \mathbf{W}^{(k)} * \mathbf{I}^{(k)} \quad (\text{Ec. 5.4})$$

Esta puede ser utilizada de igual manera en el caso de querer realizar la operación en lotes, como suele ser el caso. Para todos los casos de la Figura 43, todas las convoluciones se definen $\text{Conv2d}(C_{in}, C_{out})$. El siguiente componente presente en las capas convolucionales son las normalizaciones en *batches* [47], siendo éste otro método de normalización aplicado a redes especialmente profundas, y es utilizado para poder mejorar la capacidad de entrenamiento de las mismas. Esta normalización se basa en los valores dados por un lote específico, es diferente a la realizada en la etapa de preprocesamiento, que se realiza en base a los valores de los datos, la idea detrás de esta transformación es que una de las razones por las cuales los modelos profundos son complicados de entrenar es que la distribución de las entradas de cada capa cambia a medida que se entrena, lo mismo sucede con los parámetros de las capas anteriores. Luego de la normalización se agrega una función de activación *ReLU* (Ec. 2.2), cada capa convolucional tiene dos bloques compuestos por estos 3 componentes descriptos, un componente convolucional, una normalización por lote y una función de activación *ReLU*, al final de cada capa se encuentra una capa de *pooling* máximo, como se mencionó anteriormente, ésta permite que la representación se vuelva invariante a las traslaciones pequeñas en la entrada de la capa. La capa de *pooling* realiza esta acción sobre las dos dimensiones de altura y ancho de la imagen, serían las ultimas dos de la salida de la ecuación (Ec. 5.4) con un tamaño de la ventana para realizar la operación de 2×2 . La operación de *MaxPool2d*, teniendo como entrada $\mathcal{I} \in \mathbb{R}^{C_{out} \times H_{out} \times W_{out}}$ y un kernel de tamaño $k_H \times k_W$, se puede definir como:

$$\begin{aligned} p_{C_j, h, w} &= \max_{m=0, \dots, k_H-1} \max_{n=0, \dots, k_W-1} i_{C_j, k_H \cdot h + m, k_W \cdot w + n} \\ \forall C_j &= 1 \dots C_{out}, h = 1 \dots H_{out}, w = 1 \dots W_{out} \end{aligned} \quad (\text{Ec. 5.5})$$

La salida de este bloque es $\mathcal{P} \in \mathbb{R}^{C_{out} \times H_{pool_out} \times W_{pool_out}}$, definiendo las dimensiones de salida como:

$$\begin{aligned} H_{pool_out} &= \frac{H_{out} - (k_H - 1)}{k_H} + 1 \\ W_{pool_out} &= \frac{W_{out} - (k_W - 1)}{k_W} + 1 \end{aligned} \quad (\text{Ec. 5.6})$$

Finalmente, el último componente del bloque de la capa convolucional es una capa de *Dropout* [86] de 2 dimensiones, una técnica muy utilizada de regularización para diferentes tipos de redes. Lo

que esencialmente realiza esta capa con una cierta probabilidad es llevar el valor de salida de una unidad de una capa a cero, de ahí su nombre, enmascarar la salida y decidir aleatoriamente cuáles valores “abandonar”.

Esta técnica se puede interpretar como que entrena un ensamblado de todas las sub-redes que se forman removiendo las unidades enmascaradas en la capa sobre la cual se le aplica.

Este enmascaramiento no se realiza a la hora de la validación o el testeo del funcionamiento del modelo. Debido a que la red es un conjunto de transformaciones lineales y no lineales, se puede remover el valor de una unidad de un componente multiplicando su salida por cero.

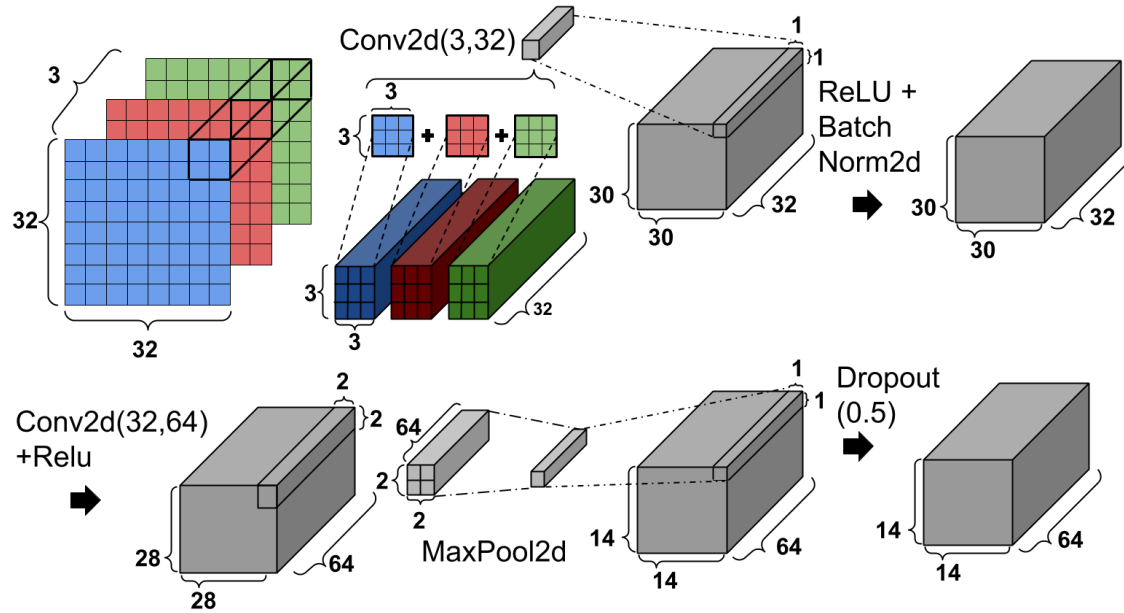


Figura 44: Diagrama del bloque de la capa convolucional 1. El resto de las capas convolucionales son idénticas en sus bloques pero con diferentes parámetros en algunos de ellos.

El bloque con la capa completamente conectada es la encargada de la tarea de clasificar cuál de las 10 categorías del set es la imagen ingresada. Los bloques con las capas convolucionales tienen las funciones de ser un *feature map*, encargado de aprender cuales son las características más importantes para poder reconocer los diferentes objetos que representa cada una de las clases a identificar. La salida de la capa completamente conectada es pasada a través de una función de *softmax* [6], previamente definida en la ecuación (Ec. 2.5), para que la salida sea un conjunto de probabilidades para poder realizar la tarea de clasificación de manera exitosa.

Luego de la definición de esta CNN de referencia, se integra parte de ésta con las arquitecturas de *Tensor Ring* y *Tensor Train* descritas anteriormente en este trabajo.

El concepto ya fue usado en los datos previamente testeados, la parte de la arquitectura modelada basándose en las descomposiciones tensoriales funciona en los ejemplos mencionados como la parte clasificadora de la red.

De una manera análoga, los bloques de la capa convolucional es un *feature map* semejante al definido en la sección anterior en la ecuación (Ec. 4.5), teniendo en cuenta este precedente es natural realizar un intercambio entre los componentes de la red, reemplazar el bloque de la capa completamente conectada y las redes tensoriales que funcionan como el componente clasificador de la red, obviamente sin el componente de *feature map*, ya que en este caso sería redundante. Un

diagrama simple explica a continuación como quedarían las redes con un componente convolucional integrado con el componente de las arquitecturas con descomposiciones tensoriales:

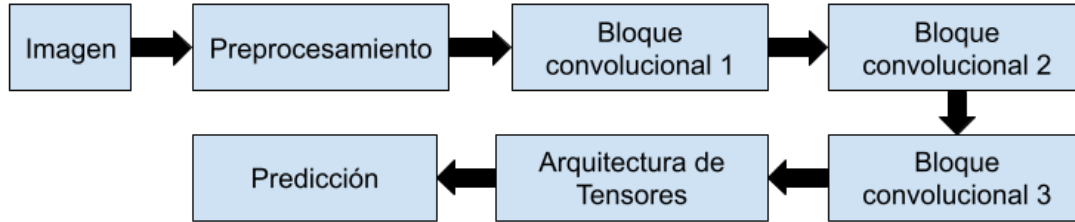


Figura 45: Red convolucional integrada con la arquitectura de tensores. Se realizaron experimentos usando tanto la red *Tensor Ring* compartida como la red *Tensor Train* compartida.

5.3.1. IMDB

Para la base de datos *IMDB*, se empezó experimentando con una red *Vanilla RNN*, utilizada en los ensayos previamente mencionados, aunque en este caso no se logró superar una barrera de rendimiento que se pueda considerar aceptable. La red realiza una clasificación con un 50 % de precisión, semejante a realizar un lanzamiento de una moneda al azar, mientras que, con las arquitecturas propuestas, el rendimiento era marginalmente mejor pero sufría del fenómeno de *overfitting*, realizaba una mejor tarea en el conjunto de validación que en el de testeo, esto suele significar que la red está sobre-ajustando sus parámetros a un conjunto particular, basándose en las experiencias anteriores con el mismo modelo, en el caso de las imágenes y el tamaño de las capas, se infiere que el sobreajuste viene principalmente de la capa de *embedding*. Una manera de mejorar el rendimiento de la red, que sufría de *overfitting*, fue la introducir una técnica anteriormente utilizada este trabajo en la CNN, *Dropout* [86], ésta es muy utilizada para regularizar diferentes tipos de redes. Su primer uso fue en redes completamente conectadas, pero actualmente su uso es amplio y en este caso se utilizó entre la capa de embedding y la red con arquitecturas de tensor. Aplicarlo a la hora de la capa donde se transforman los tokens en vectores que la red puede calcular, puede interpretarse no sólo como un ensamblado de diferentes *embedding* pre-entrenados sino también como diferentes variaciones de los conjuntos de entrenamiento. Es importante remarcar que todos estos datos y modelos “diferentes” están altamente correlacionados entre sí y no pueden interpretarse como independientes, aún así es un método que demostró ser bastante efectivo siendo computacionalmente barato.

En un intento por mejorar los resultados obtenidos con el conjunto de críticas y experimentar con otro tipo de arquitecturas utilizadas actualmente, se recurrió a utilizar una red CNN para mejorar el rendimiento obtenido. Teniendo como base el trabajo de [55], se utilizó una CNN para reducir la dimensionalidad de la entrada a la red. Tradicionalmente, las CNN son utilizadas en imágenes y están compuestas generalmente por múltiples capas convolucionales. Estas capas contienen un filtro con un kernel cuyos valores son aprendidos por la red y son los que realizan la convolución de esas imágenes para transfórmalos en feature maps de menor dimensionalidad preservando las características más importantes de la imagen. Esta parte de la capa funciona como un extractor de las características relevantes de la entrada para la red, estos filtros suelen ser de dos o tres dimensiones dependiendo del tipo de imagen y el diseño de la red. El texto no puede ser fácilmente interpretado como naturalmente multidimensional, es más natural pensarlo como un vector unidimensional de palabras. Luego de ser transformado a través de los *embedding* se convierte en una matriz de 2

dimensiones, en una dimensión cada palabra en el orden dado por el orden por el texto y en otra vectores representando cada palabra, con los valores pre-entrenados dados. Teniendo esto en cuenta, se puede pensar en un filtro de dos dimensiones que tenga como primera dimensión el tamaño del *embedding* elegido y como segunda dimensión una cantidad arbitraria de palabras en forma de n-gramas, así queda definido un *kernel* de tamaño $\mathbb{R}^{n \times e}$. Las capas convolucionales, en este caso de 2 dimensiones, tienen además definido un canal de entrada que se relaciona con la cantidad de canales que tiene cada entrada, en el caso de una imagen definida en RGB son tres, en el caso del texto será 1. El canal de salida es el tamaño de la salida del tensor. Puede ser interpretado como la cantidad de filtros que tiene la capa convolucional.

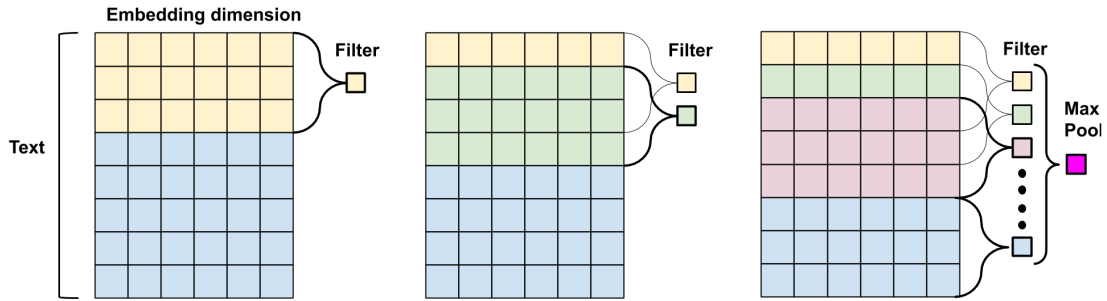


Figura 46: Ejemplo de cómo se aplica un filtro para un n-grama, de derecha a izquierda para cada agrupación de 3 palabras. Finalmente se aplica un max pooling sobre el total de los resultados del filtro. Esta operatoria es realizada para cada filtro y para tamaño de n-grama.

A la hora de implementar esta idea se decidió utilizar n-gramas de tamaño 3, 4 y 5, cada uno de los filtros es de tamaño $\mathbb{R}^{3 \times 50}$, $\mathbb{R}^{4 \times 50}$ y $\mathbb{R}^{5 \times 50}$ respectivamente. El canal de entrada cómo se mencionó es de tamaño 1 y la cantidad de estos filtros es de 100 para cada uno de los n-gramas. Entonces, cada uno de éstos es una capa convencional de 2 dimensiones con filtros de diferente tamaño, entonces por ejemplo la capa con un kernel de 3×50 , en la práctica agrupará de a 3 palabras para cada posición del texto y aplicará 100 filtros de este tamaño para cada agrupación, teniendo como resultado una matriz donde una dimensión es el tamaño del texto agrupado de a 3 palabras, por ejemplo para un texto de 200 palabras habría 198 agrupaciones. La otra dimensión de la salida sería la cantidad de filtros, en este caso de 100, el concepto detrás de esto es que cada filtro aprenda una característica, aumentando así la cantidad de información relevante extraída. Luego de la capa convolución se aplica por convención una función de activación *ReLU*, por razones ya mencionadas en el trabajo. Luego una capa de pooling, que como también se refirió anteriormente, se aplica con el objetivo de que la representación se vuelva invariante a las traslaciones pequeñas en la entrada de la capa. Este último es un concepto relevante a la hora de utilizar para reconocer objetos en imágenes, sin importar el lugar o la posición en la que se encuentra, también es muy útil si se lo aplica a la tarea de clasificar el sentimiento de un texto, poder identificar frases o palabras que caracterizan a una crítica positiva de una negativa restándole importancia a la ubicación en el texto o si ésta se encuentra entre otras que resultan irrelevantes para esta tarea. En este caso se utilizó una capa de *max pooling* aplicada sobre la dimensión de las agrupaciones. La concepción detrás de esto es que el valor más grande es el más importante a la hora de evaluar una entrada y éste es determinado por el n-grama más relevante. La tarea de la red es aprender cual es ese n-grama y ser capaz de identificarlo para poder ser eficaz a la hora de poder realizar su finalidad más efectivamente. El resultado final de estas 3 capas que son el componente fundamental de una red CNN son los 300 *features* que éstas evalúan considerándolos los más relevantes para determinar el sentimiento del

texto.

La capa convolucional consta de los tres componentes detallados anteriormente, los vectores producidos luego del *embedding*, son procesados por los filtros convolucionales de diferentes tamaños, por una capa de activación (ReLU), para finalmente realizar una función de *pooling* sobre la dimensión del texto. Estos *features* resultantes son los que luego son utilizados por una última capa que se encarga de realizar la tarea de clasificar el sentimiento de la crítica. Ésta última capa en los trabajos de referencia, suele ser utilizado a la hora de clasificar otro tipo de entradas como imágenes, una capa completamente conectada. Estos *features* son además regularizados antes de pasar por el último componente de la red utilizando una técnica descrita anteriormente llamada *dropout*, con el objetivo de que éste no sufra el fenómeno ya mencionado denominado *overfitting*. Para experimentar con este diseño utilizando las arquitecturas propuestas, se decidió reemplazar la capa completamente conectada por la red TT o alternativamente la red TR.

Un detalle importante a notar en la implementación de la red *Tensor Ring* (ver Figura 27), para todos los resultados posteriores y las consecuentes conclusiones del trabajo, es remarcar que el rango $R_{\tau+2}$ se fijó para todos los experimentos en un valor igual a $R_{\tau+2} = R' = 10$. Este valor de hiperparámetro fue seteado en base a una pequeña serie de ensayos sobre la base *MNIST*, en principio se experimentó con un valor de $R' = R$, pero esto impacta de manera considerable en el rendimiento de la red, especialmente respecto a la cantidad de operaciones en la ejecución de *feedforward* (Ec. 4.50), se vuelve prohibitivo el tiempo que demanda sin tener un impacto positivo en su rendimiento, también se vuelve oneroso en la cantidad de parámetros de la red. Teniendo en cuenta que, si el valor $R_{\tau+2} = 1$, la arquitectura *Tensor Ring* se vuelve equivalente a una red *Tensor Train*, se llegó a ese valor debido a que éste en un orden similar a los valores experimentales de R , pero no se vuelve tan costoso a medida que el valor del Rango aumenta, básicamente limita el crecimiento cúbico de la cantidad de operaciones cuando el valor de $R' \ll R$.

Se realizaron pruebas con las arquitecturas propuestas con la CNN al principio de la red, pero intercambiando las redes de la parte descrita como clasificadora por una capa completamente conectada con una entrada de tamaño de $f_s \cdot n_f$, siendo $f_s = 100$ el tamaño de los filtros y $n_f = 3$ la cantidad de filtros, como se detalló anteriormente cada uno de diferente tamaño (3,4,5) para representar cada uno un tamaño de n-grama. La dimensión de salida es la cantidad de categorías a clasificar que en esta ocasión es de tamaño uno.

5.4. Detalles de implementación y selección de hiperparámetros

A la hora de realizar los diferentes experimentos que son parte de este trabajo, hay una gran cantidad de hiperparámetros, cuyos valores deben ser determinados en las diferentes arquitecturas para cada uno de éstos. Varios ya fueron repetidos múltiples veces durante este trabajo, pero es importante notar que los valores que definen la instancia de una red para cada uno de los experimentos son R el rango de los tensores, m la cantidad de *features* definidos para cada uno de las entradas para el tensor, τ la cantidad de pasos que tiene la red, que son al mismo tiempo la cantidad de divisiones (partidas de diferentes maneras) para la entrada a la red. En base a esos 3 hiperparámetros se calcula el tamaño de la red, que en algunos casos, como para las bases *IMDB* y *CIFAR-10*, tiene otros componentes que no son la arquitectura planteada por este trabajo, pero aún así afectan de manera directa a la cantidad total de parámetros (Ec. 4.56) (Ec. 4.55). La correspondencia entre la cantidad de parámetros y la precisión va a ser uno de los puntos de análisis en el trabajo.

Por otro lado, se determinan los parámetros de aprendizaje de la red, éstos fueron validados en base a experimentos para determinar los mejores resultados prácticos luego de un par de iteraciones sobre los mismos pero se mantuvieron constantes a través de todos los experimentos realizados.

La tasa de aprendizaje se mantuvo en $1 \cdot 10^{-4}$, el tamaño del *batch* es de 256. El algoritmo de optimización utilizado para los experimentos es *Adam* [56], con cambios en la penalización de L_2 propuestos por el trabajo de *Loshchilov 2017*[68]. La pérdida es calculada con el algoritmo previamente mencionado llamado *negative log-likelihood* (Ec. 2.6).

La base de datos *IMDB* presenta una diferencia con respecto al resto de los experimentos: éste tiene una sola clase, debido a que su objetivo es el de determinar si los sentimientos de la crítica son positivos o negativos. En este caso se decidió experimentar con una función diferente para calcular la pérdida para cada *batch*, en este caso se utilizó *Binary Cross Entropy with Logit Loss*. Lo primero a aclarar es que la función de *Cross Entropy* es equivalente a calcular con la función *negative log-likelihood* luego de aplicar *softmax*, como en el resto de los experimentos en este trabajo. La diferencia con *Binary Cross Entropy* es que está integrada con una función Sigmoide σ (Ec. 2.3). El objetivo de esta operación es que devuelve un valor entre 0 y 1 pero con una distribución que puede llevar todos los valores cerca de alguno de esos dos límites. Esto intuitivamente facilita el objetivo de clasificar en una categoría binaria donde se debe elegir entre uno de esos dos valores.

5.5. Resultados

5.5.1. MNIST

Los primeros experimentos realizados fueron sobre la base *MNIST* [66]. Los parámetros de evaluación establecidos para estos se probaron con dos valores diferentes de m , 32 y 64. Se utilizaron varias combinaciones de estos dos valores con diferentes valores de R , para poder comparar ambas arquitecturas en dos variables diferentes, una como se comporta con el mismo m y cambiando el R en valores equivalentes. La otra como se comporta cuando manteniendo el m constante ambas tienen la misma cantidad de parámetros, viendo no sólo la efectividad de las redes, sino comparándolos punto a punto para poder sacar conclusiones acerca del comportamiento de ambas redes.

Otros hiperparámetros definidos para estos experimentos pero que se mantienen constantes para el resto, como se explica en la Figura 35, se eligió un valor de $\tau = 16$, dividiendo cada una de las imágenes de entrada en 16 divisiones de 7x7 píxeles.

Se realizaron pruebas sobre las 3 arquitecturas propuestas previamente en este trabajo (*paralela*, *serial* y *compartida*). Los resultados que se muestran a continuación son para el caso de la *compartida* en todos los casos. Los resultados para ambas, la *serial* y especialmente la *paralelizada*, no fueron buenos en los resultados preliminares, por eso se decidió no profundizar las pruebas con estas arquitecturas.

Los resultados de los experimentos, comparando la precisión con respecto al rango R para ambas arquitecturas se muestran en las Figuras 47 y 48.

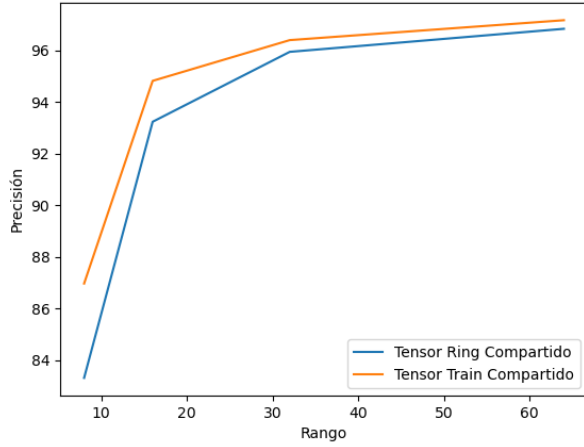


Figura 47: Interpolación de las mejores resultados sobre el conjunto de los test respecto del rango para *MNIST* con $m = 32$.

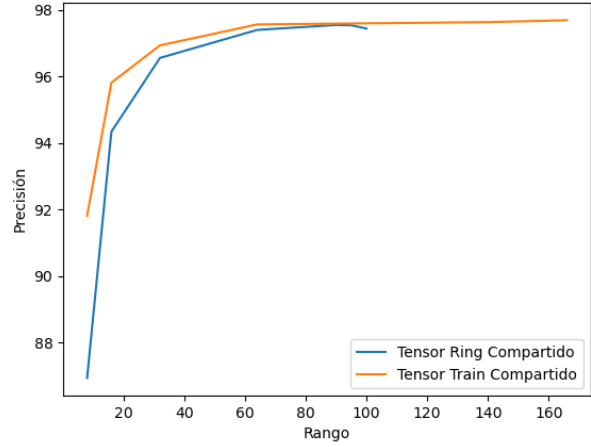


Figura 48: Interpolación de las mejores resultados sobre el conjunto de los test respecto del rango de *MNIST* con $m = 64$.

Los resultados de los experimentos comparando la precisión con respecto a la cantidad de parámetros para ambas arquitecturas se muestran en las Figuras 49 y 50.

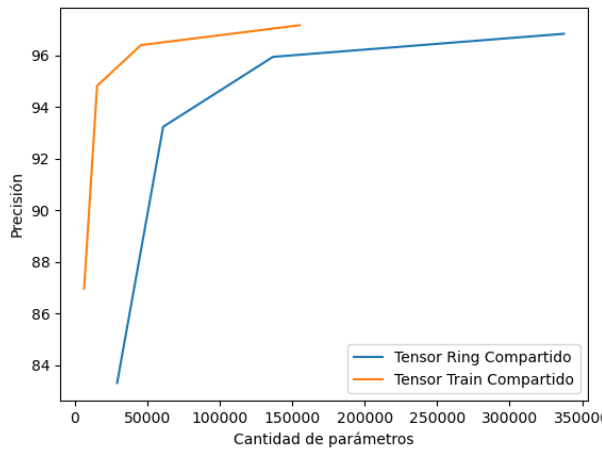


Figura 49: Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros de *MNIST* con $m = 32$.

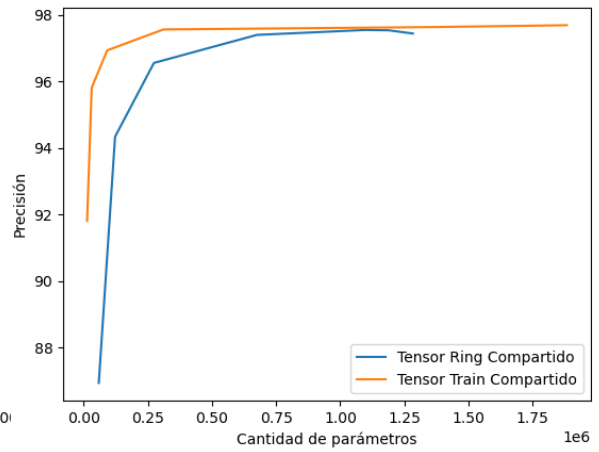


Figura 50: Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros de *MNIST* con $m = 64$.

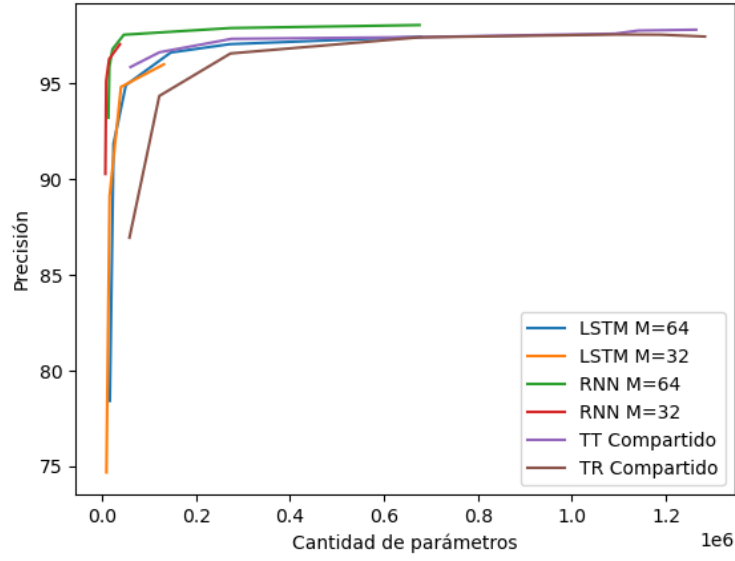


Figura 51: Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros para cada una de las redes con *MNIST*.

m	Red	R	Precisión	parámetros	Red	R	Precisión	parámetros	τ
64	TR	8	92.32 %	58496	TT	25	96.29 %	60800	16
64		16	95.94 %	121984		38	96.97 %	122368	16
64		32	96.93 %	273536		60	97.49 %	275840	16
64		64	97.65 %	674944		97	97.68 %	673664	16
64		90	97.87 %	1097600		125	97.72 %	1091200	16
64		95	97.68 %	1188800		128	97.95 %	1141888	16
64		100	97.78 %	1283200		135	97.89 %	1264640	16
64	TT	8	92.92 %	12928	TT	64	97.84 %	310400	16
64	TT	16	96.5 %	30848	TT	140	97.71 %	1356160	16
64	TT	32	97.17 %	91264	TT	166	97.99 %	1883648	16

Tabla 6: Mejor Rendimiento de las diferentes redes con arquitectura *compartida* para *MNIST* con $m=64$

m	Red	R	Precisión	parámetros	Red	R	Precisión	parámetros	τ
32	TR	8	88.74 %	29248	TT	8	91.18 %	6464	16
32		16	95.08 %	60992		16	95.43 %	15424	16
32		32	96.38 %	136768		32	96.8 %	45632	16
32		64	97.38 %	337472		64	97.38 %	155200	16

Tabla 7: Mejor Rendimiento de las diferentes redes con arquitectura *compartida* para *MNIST* con $m=32$

Observando los experimentos realizados para la base *MNIST* [66], se pueden obtener varias conclusiones. Los resultados de los experimentos comparando la precisión con respecto al rango R , mostrados en las Figuras 47 y 48, demuestran que con rangos pequeños (8 y 16), la red *Tensor Train* (TT) clasifica mejor que la red *Tensor Ring* (TR). Se puede deducir de esto que la arquitectura TR con rango menor tiene menor poder expresivo que para la arquitectura TT, pero a medida que aumenta el rango, el rendimiento de ambas se vuelve equivalente.

Las Figuras 49 y 50 muestran los mismos resultados pero en base a la cantidad de parámetros de las redes. Teniendo en cuenta las ecuaciones (Ec. 4.55) y (Ec. 4.56) se puede ver que el coeficiente cuadrático de la red TR para el número de parámetros es claramente mayor al de la red TT. Este fenómeno se puede observar directamente en la Figura 57, también reflejado en estas dos imágenes. Para $m = 32$ con rango 8 para la arquitectura TT, la cantidad de parámetros es de 6.464, mientras que para la otra es de 29.248, aún así la primera funciona mejor que la otra, esto trae a la luz la misma idea que con el rango, esta arquitectura necesita de mayor rango, que implica mayor cantidad de parámetros para igualar el mismo rendimiento. Pero para llegar a un rango en el cual el rendimiento es similar, la cantidad de parámetros debe ser mucho mayor.

Otra observación interesante de la Figura 50, es que a partir de cierta cantidad de parámetros el modelo llega a un límite en su capacidad de predicción, no puede “aprender” más. Si se continúa aumentando la cantidad de parámetros, la red corre riesgo de sufrir del fenómeno de *overfitting* que como ya se describió puede llevar a empeorar el funcionamiento de la red.

Con respecto a los experimentos realizados con diferentes *features maps*, es consistente para ambas redes afirmar que con un valor de m más grande, el funcionamiento es mejor. Principalmente con $m = 64$ el rendimiento es mejor que con $m = 32$, esto es verdad para todos los rangos para las dos arquitecturas. Esto es consistente con la teoría que, mientras más *features* o conocimiento se le da a la red para cada imagen, ésta puede realizar su tarea de clasificar de una manera más certera. Es importante notar que los resultados de los experimentos para rangos más pequeños tienen en general un nivel mayor de varianza que los rangos más altos, esto nos lleva a que con menor cantidad de parámetros es más dependiente de la inicialización de los parámetros.

Este comportamiento con un límite en la capacidad de rendimiento se puede observar en ambas arquitecturas y con diferentes tamaños de *features maps*, se torna consistente en los experimentos realizados. Además es importante tener en cuenta el tiempo derivado de la cantidad de operaciones tanto en la red TT (Ec. 4.47) como en la red TR (Ec. 4.50), que crece de manera cuadrática respecto del rango en el primer caso y de manera cúbica para el segundo caso. Cada vez que el tensor se vuelve más “ancho” de alguna manera aumentando su volumen de datos, se vuelve más costoso de almacenar, y se aumenta aún más la penalidad en las operaciones para calcular el tensor. Esto afecta no sólo el tiempo de *feedforward* a la hora de utilizar la red en un ambiente productivo, también

se incrementa de manera importante el tiempo y los recursos para entrenar esta red. Examinando esta evidencia se infiere que es importante encontrar el equilibrio entre el tamaño de la red y el rendimiento, esto se transforma en una decisión que es compleja de tomar, pero siempre se tiene que tener en cuenta todas estas variables a la hora de elegir no sólo una arquitectura, sino sus parámetros conociendo sus ventajas y desventajas. Teniendo en cuenta el equilibrio descripto entre tamaño, rendimiento y tiempo.

Analizando la Figura 51, en ésta se visualizan el rendimiento de 6 tipos de redes con experimentos realizados con diferentes hiperparámetros, en cada una de las líneas la constante es el tamaño de los *Features maps* m y lo que varía es el rango R , aunque el gráfico se ve la relación del rendimiento en base a la cantidad de parámetros. Como se mencionó en la sección previa, además de los ensayos realizados para ambas arquitecturas propuestas, se realizaron pruebas con una red LSTM (ver Sección A.2) y una red que arriba definimos como *Vanilla Recurrent Neural Network* (ver Sección 2.4), de ahora en más denominada RNN. Estas dos tienen comportamientos similares a los analizados previamente, a medida que aumenta el rango, y por lo tanto la cantidad de parámetros, se encuentra un límite asintótico en el desempeño. A diferencia de las otras dos, no se perciben disparidad en su comportamiento con diferentes hiperparámetros m , en ambos casos son casi calcadas. También se discierne que sufren la misma conducta ante el aumento de rango, se llega a un límite asintótico y con valores que se pueden considerar equivalentes entre ellos. En este contexto es interesante advertir que a pesar que la red RNN, la cual puede llegar a considerarse como la más simple también es la que más productiva, se deduce que puede deberse a la complejidad del problema y las propiedades singulares de la base de datos.

El objetivo principal de los ensayos realizados con *MNIST* fue el de validar que las redes pueden cumplir con el objetivo de clasificar las imágenes de números escritos manualmente en escala de grises (ver Figura 33), que componen la base de datos, de manera efectiva. Comparando los resultados obtenidos, no se encuentra a la altura del estado del arte para esta tarea (ver Tabla 5), pero se encuentran claramente arriba de un nivel para el cual se puede afirmar que cumplen con creces la meta planteada para el experimento.

En la comparación entre ambas arquitecturas se puede concluir que con menor cantidad de parámetros la red TT funciona mejor, pero a medida que el rango aumenta, con la misma cantidad de parámetros las dos performan de manera equivalente.

5.5.2. Fashion-MNIST

Para los experimentos ensayados sobre la base *Fashion-MNIST* [95], los parámetros establecidos se probaron con dos valores diferentes de m , 32 y 64, al igual que para el caso anterior. De la misma manera, se utilizaron varias combinaciones con diferentes valores de R . Al igual que para el set previo, para poder comparar la red *Tensor Train* con la red *Tensor Ring* en dos variables distintas, cuál es la diferencia entre los dos valores de m seleccionados variando el rango R en valores equivalentes, notando las diferencias entre las redes con el mismo m y entre la misma red con la diferencia entre el tamaño de los *features maps*.

El siguiente gráfico muestra como se comportan las redes variando la cantidad de parámetros mientras se mantienen constante el valor de m , viendo no sólo la efectividad de las redes, sino comparándolos punto a punto para poder sacar conclusiones acerca del rendimiento de ambas redes. Se eligió un valor de $\tau = 16$, dividiendo cada una de las imágenes de entrada en 16 divisiones de 7x7 píxeles cada una, al igual que el experimento previo.

Los resultados de los experimentos comparando la precisión con respecto al rango R para ambas arquitecturas son:

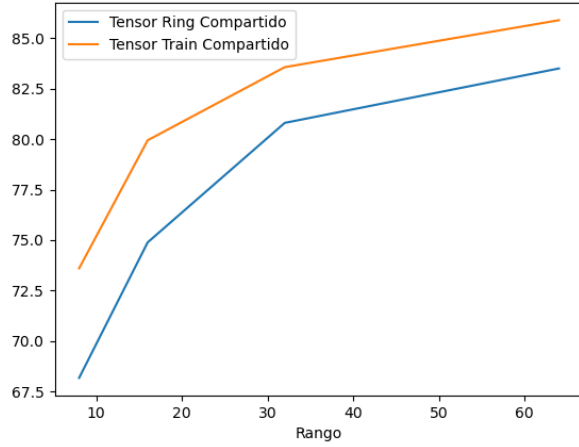


Figura 52: Interpolación de las mejores resultados sobre el conjunto de los test respecto del rango de *Fashion-MNIST* con $m = 32$.

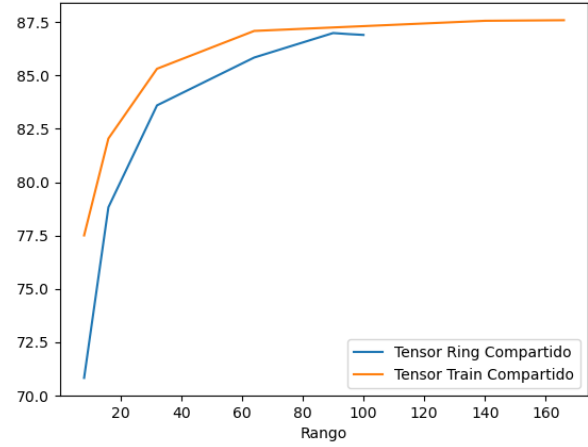


Figura 53: Interpolación de las mejores resultados sobre el conjunto de los test respecto del rango de *Fashion-MNIST* con $m = 64$.

Los resultados de los experimentos comparando la precisión con respecto a la cantidad de parámetros para ambas arquitecturas son:

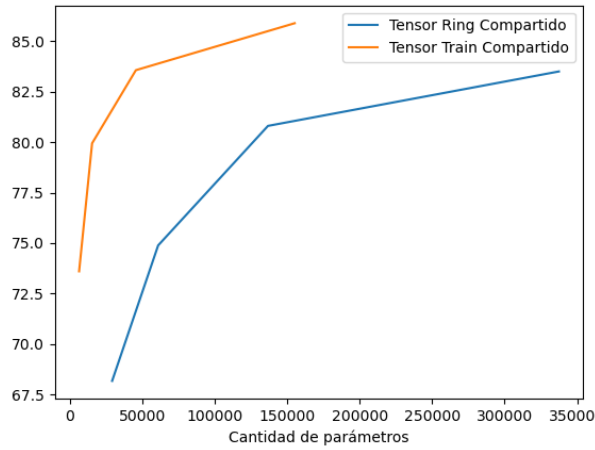


Figura 54: Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros de *Fashion-MNIST* con $m = 32$.

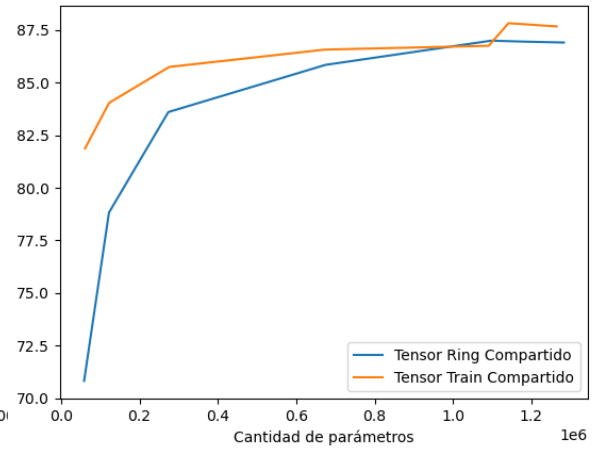


Figura 55: Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros de *Fashion-MNIST* con $m = 64$.

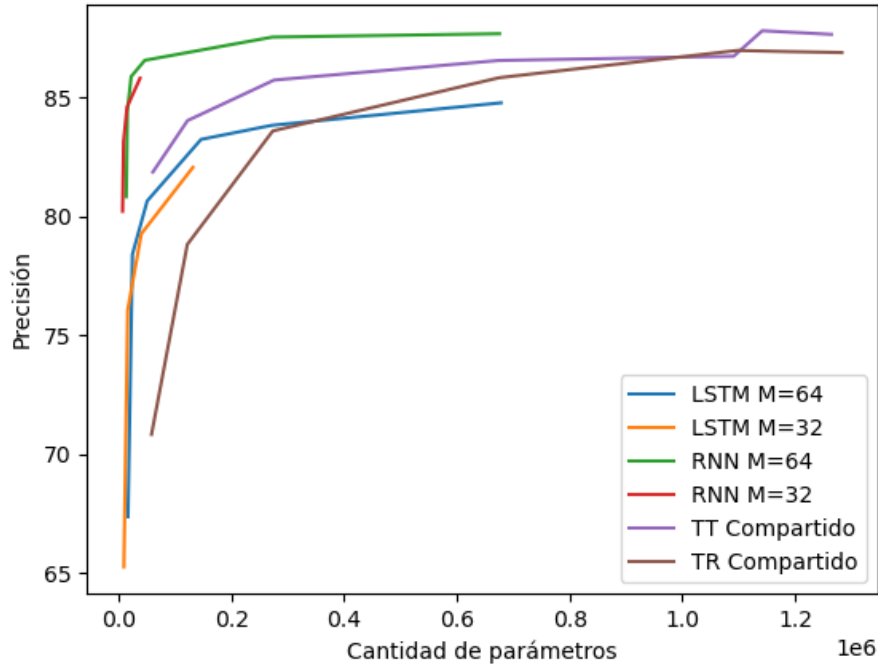


Figura 56: Interpolación de las mejores resultados sobre el conjunto de los test respecto de la cantidad de parámetros para cada una de las redes con *Fashion-MNIST*.

m	Red	R	Precisión	parámetros	Red	R	Precisión	parámetros	τ
64	TR	8	74.11 %	58496	TT	25	83.12 %	60800	16
64		16	80.37 %	121984		38	84.49 %	122368	16
64		32	84.3 %	273536		60	86.63 %	275840	16
64		64	86.25 %	674944		97	86.9 %	673664	16
64		90	87.19 %	1097600		125	87.0 %	1091200	16
64		95	87.42 %	1188800		128	88.43 %	1141888	16
64		100	87.68 %	1283200		135	87.83 %	1264640	16
64	TT	8	79.06 %	12928	TT	64	87.78 %	310400	16
64	TT	16	83.54 %	30848	TT	140	88.14 %	1356160	16
64	TT	32	86.22 %	91264	TT	166	88.14 %	1883648	16

Tabla 8: Mejor Rendimiento de las diferentes redes con arquitectura *compartida* para *Fashion-MNIST* con $m=64$

m	Red	R	Precisión	parámetros	Red	R	Precisión	parámetros	τ
32	TR	8	72.98 %	29248	TT	8	76.59 %	6464	16
32		16	77.17 %	60992		16	81.8 %	15424	16
32		32	81.78 %	136768		32	85.12 %	45632	16
32		64	84.06 %	337472		64	86.91 %	155200	16

Tabla 9: Mejor Rendimiento de las diferentes redes con arquitectura *compartida* para *Fashion-MNIST* con $m=32$

El próximo gráfico muestra como evoluciona el número de parámetros con respecto al rango R las redes con el tamaño de los *features maps* m . Éste describe la relación detallada por las ecuaciones para la cantidad de parámetros para la red *Tensor Ring* (Ec. 4.55) y *Tensor Train* (Ec. 4.56).

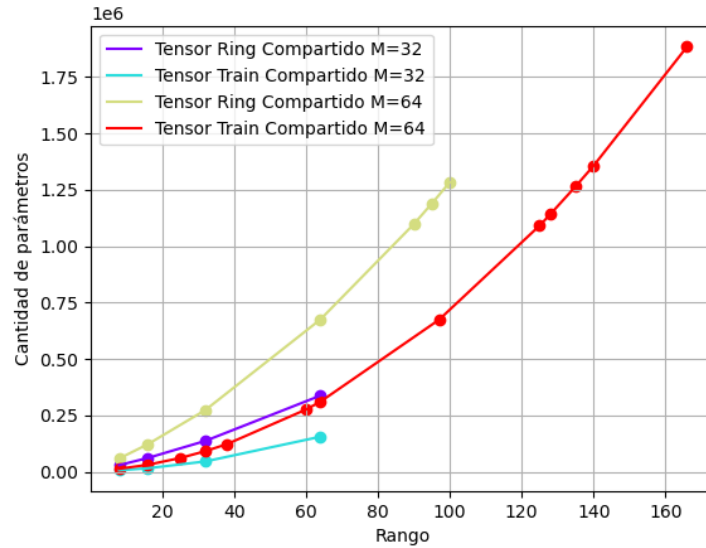


Figura 57: Parámetros para los ensayos para *MNIST* y *Fashion-MNIST*

En los ensayos para *Fashion-MNIST* se puede apreciar la diferencia con los realizados con *MNIST* respecto del rendimiento de las redes para un problema análogo. Esto es debido a que la complejidad del problema es mayor. Para el caso con $m = 32$, para el caso de una red TT, ver Figura 47 y una red TR, ver Figura 52, se advierte que su desempeño, a medida que aumenta el rango, no tiene el mismo desarrollo, y se puede atribuir a la dificultad del problema, debido a que las arquitecturas son equivalentes en ambos casos. Para los ensayos con $m = 64$, se advierte un comportamiento muy similar con los mismos realizados para el *MNIST*. A medida que crece el rango R , ambas convergen a un límite asintótico de rendimiento, notablemente menor que el caso del set anterior.

La mejor corrida para TT en *MNIST* es de 97,95 % y para TR 97,87 %, mientras que para *Fashion-MNIST* es de 87,68 % para TR y de 88,43 % para TT. Lo que se puede concluir de estos resultados es que se cumplió la hipótesis planteada, que es un problema mucho más complejo debido a que sus rendimientos fueron consistentemente mejores utilizando para las dos redes las mismas

configuraciones e hiperparámetros para ambos conjuntos de datos.

Examinando las Figuras 54 y 55, se puede repetir el mismo análisis que para el caso de *MNIST*, donde el tamaño de los *Features Maps* igual a 32. Se infiere que el rendimiento de la red TT es mejor para una cantidad igual de parámetros y, para el mismo rango, la cantidad de parámetros para una red TR es mucho mayor. Para el caso con $m = 64$, el gráfico es muy similar al de los experimentos con el otro set: la performance mejora notablemente a medida que aumenta la cantidad de parámetros, luego se estabiliza y el rendimiento se vuelve independiente al número de parámetros, a medida que estos continúan aumentando. Como es el caso para los experimentos con *MNIST* y análogamente con el análisis realizado respecto a la cantidad de parámetros, luego de llegar a una cierta suma, el modelo parece arribar a una cota en la capacidad de clasificar, que, aunque siga subiendo, no parece poder aumentar su capacidad de 'aprender'.

Analizando los ensayos en base a los diferentes valores de m , éste repite el mismo patrón ya detectado, para ambas redes los resultados son mejores con $m = 64$ que con $m = 32$. Esto sucede independientemente al variar el tamaño del rango R , con valores 8, 16, 32 y 64. Debido a la correlación de esta variable con la cantidad de parámetros (Ec. 4.55) (Ec. 4.56), mejora su tarea con un valor de m más alto.

Al examinar la Figura 56, las conclusiones son similares a las encontradas para la base *MNIST*. La red LSTM (ver Sección A.2) y una red *Vanilla RNN* (ver Sección 2.4) sufren el mismo fenómeno con diferentes valores de m . El comportamiento es similar, ambos llegan a su límite de rendimiento y luego se mantienen estables, mientras crece el rango para el caso de estas redes el tamaño del estado oculto. En este caso se replica la misma conducta que se observa para el otro set, la red *Vanilla RNN* realiza una mejor tarea para clasificar las imágenes que la red LSTM, aunque ésta última se considera más compleja y potente que la primera. Esto puede deberse a que no sólo el problema se puede considerar como "simple", sino que las características alimentadas a la red no son capaces de darle suficiente información capaz de detectar patrones más complejos. Otro factor que influye en este comportamiento, es que los experimentos son valiosos a la hora de validar que las redes son capaces de realizar la tarea de predecir y clasificar de manera efectiva imágenes en diferentes categorías, pero siguen sin ser como se mencionó anteriormente los problemas para los cuales las redes recurrentes son las más efectivas.

Considerando la Figura 56, junto a la Figura 51, su análoga para los ensayos de *MNIST*, podemos advertir un patrón ya identificado para las arquitecturas propuestas, pero se repite para el caso de las redes LSTM y *Vanilla RNN*, a medida que aumenta el tamaño del vector de *Features maps* m también lo hace el funcionamiento de todas las redes. Esto refuerza el corolario, el rendimiento con un valor de $m = 64$ es mejor que con un valor de $m = 32$ resulta que las redes en general, clasifican mejor si se les alimenta con más características para cada una de las entradas.

Evaluando los resultados para ambas redes, de la misma manera que para *MNIST*, la red TT performa mejor con menor rango y con un *feature map* de tamaño más pequeño, pero a medida que aumenta el rango los resultados de la red TR performa de una manera similar y ambas llegan a un límite asintótico en su funcionamiento. En este caso el objetivo planteado también puede considerarse como cumplido, se ensayó con una base de datos más compleja y ambas redes demostraron tener la capacidad de poder realizar la tarea de manera efectiva. De la misma manera los resultados obtenidos no se encuentran en el orden del estado del arte (ver Tabla 5) para este conjunto de datos, pero cumple el objetivo propuesto satisfactoriamente.

5.5.3. CIFAR-10

Para la base *CIFAR-10* se ensayaron con tres tipos de redes diferentes, utilizando tres configuraciones distintas tanto para la arquitectura *Tensor Ring* (ver Figura 45) como para la arquitectura

Tensor Train (ver Figura 45). El objetivo era probar estas dos utilizando varios valores diferentes de m , con distintos rangos para las dos redes y así poder realizar la comparación entre ellas con cantidades similares de parámetros. Para el caso de la red CNN (ver Figura 43) sólo se realizaron experimentos con un sólo conjunto de hiperparámetros con el propósito de establecer una cota de referencia en el rendimiento de una red similar para este conjunto de datos.

Red	Precisión	m	R	Cantidad de parámetros	τ
<i>CNN</i>	83.81 %	-	-	1258346	-
CNN-TT	83.55 %	32	38	1219296	4
	83.49 %	64	38	1311680	4
	83.59 %	128	38	1495680	4
CNN-TR	83.16 %	32	16	1219104	4
	83.83 %	64	16	1311296	4
	83.11 %	128	16	1495680	4

Tabla 10: Mejor Rendimiento de las diferentes redes para *CIFAR-10*

En la sección anterior 5.1 se mencionó que los primeros ensayos, utilizando el preprocesamiento y las arquitecturas planteadas para los ensayos previos, dieron resultados magros para este conjunto de datos. Esta base de datos es aún más compleja que las utilizados anteriormente, la cantidad de categorías, el formato de las imágenes, dado que éste tiene 3 colores a diferencia de la escala de grises de *MNIST* Y *Fashion-MNIST*. Esto resultó en que el mejor rendimiento encontrado para diferentes hiperparámetros para la red TT sea de alrededor 50 % de precisión. Aunque cumple con los requerimientos mínimos para afirmar que la red realiza la tarea de manera efectiva, se intentó con la meta no sólo de lograr un mejor rendimiento, sino también poder experimentar con otro tipo de redes y diseño. Se planteó una red que se denominó CNN-TT (ver Figura 45) y otra denominada CNN-TR (ver Figura 45), ambas descritas en la sección previamente mencionada. Además de estas dos redes, se plantea una tercera red (ver Figura 43) con el objetivo de obtener una red de referencia que sirva como un punto de comparación para los experimentos realizados.

Se realizaron los experimentos con los modelos antes mencionados (ver Tabla 10), para las redes CNN-TR y CNN-TT, se ejecutaron ensayos con diferentes valores de m pero con valor fijo de R , diferente entre ambas redes pero que permite que se vuelvan comparables en la cantidad de parámetros entre sí. Partiendo de las conclusiones de los experimentos con los conjuntos de datos mencionados se decidió tener un R fijo, variando el tamaño de los Features maps debido a que se identificó en los ensayos anteriores que éste es un hiperparámetro importante a la hora de mejorar el rendimiento de la red, además esta mejora tiene un menor impacto, en el tamaño y cantidad de operaciones en la red. En el caso de la red TT, ésta incrementa su cantidad de parámetros de manera lineal con respecto al valor de m (Ec. 4.56), para el rango R esta relación se vuelve cuadrática. La relación también es lineal para m en el caso de la cantidad de operaciones por cada *feedforward* (Ec. 4.47), y es cuadrática para el rango R para la misma. Para la red TR la relación también es lineal entre m y el número de parámetros (Ec. 4.55), y para el número de operaciones (Ec. 4.50), mientras que para el rango R es cuadrática y cúbica respectivamente.

Teniendo en cuenta esto, los resultados (ver Tabla 10) arrojaron bastante paridad entre ambas redes. Es importante que tanto para la red TT, como para la red TR la mejor precisión obtenida no cambia demasiado respecto al aumento del tamaño de los *Features Maps* m y por lo tanto en la cantidad de parámetros.

Otra observación interesante a notar para esta serie de experimentos es que los rendimientos de la red TT y la red TR, comparadas entre sí, son muy similares para todos los valores de m . Mirando los experimentos anteriores, tanto para *MNIST* (ver Tablas 7 y 8) como para *Fashion-MNIST* (ver Tablas 9 y 8), es apreciable la diferencia en la mejor precisión obtenida al variar el valor de m , éste no es el caso para esta serie de ensayos. Puede deberse a que como se mencionó la manera de obtener los *features* cambia significativamente, en los primeros dos casos se realiza el preprocesamiento descrito en la ecuación (Ec. 4.23), mientras que ya se mencionó este caso los vectores que alimentan a los modelos tensoriales son los que resultan del paso de las imágenes por la red CNN. Ésta diferencia en las entradas para las redes, en la complejidad, el tamaño de los componentes que realizan esta transformación y principalmente en el cambio de paradigma de cómo obtiene los *features* la secuencia de la CNN (ver Figura 43) contrastado con el de una red completamente conectada que realiza una transformación lineal, pueden explicar porque en estos ensayos los cambios en los componentes TT y TR, que como se denominó antes son el componente *clasificador* de la red, no impactan de una manera considerable a los resultados finales obtenidos. Se puede interpretar que en las primeras dos series de experimentos, la parte clasificadora cobra mayor relevancia cuando la construcción de los *features* es más simple, mientras más capacidad tengan estas redes de aprender a interpretar estos *features*, esto impactará mejorando el rendimiento a la hora de realizar dicha tarea. Mientras que en el caso donde los *features* son más complejos y ricos en información, como puede ser el caso por la salida de las CNN, en esta coyuntura, las redes tensoriales tienen menor ponderación en el resultado final debido a que, aunque aumenten su tamaño y por lo tanto su capacidad de detectar diferentes relaciones, las redes más pequeñas pueden hacerlo con una efectividad similar ya que la entrada provee la información que necesita para poder efectuar esta tarea.

Para comparar los resultados de las redes TT y TR, se realizaron ensayos con la CNN de referencia definida en la sección 5.1 (ver Figura 43), que tiene el mismo componente convolucional que las otras dos y se diferencia en que el bloque que se encarga de clasificar los *features*, es una red completamente conectada. Los experimentos con esta red arrojaron un rendimiento parecido a los anteriormente descritos en esta sección, con una cantidad de similar de parámetros, la diferencia no permite afirmar claramente que ésta desempeña mejor su tarea. Lo que permite afirmar es que las redes tensoriales no funcionan mejor que las capas completamente conectadas, en este caso 2 capas de este tipo con un rectificador *RELU* (Ec. 2.2) y un dropout entre ellas. Esto permite aseverar con mayor certeza las conclusiones previamente esbozadas en el párrafo anterior, en el caso de los modelos delineados para estos ensayos, el componente convolucional, común para todos ellos, es el que puede definirse como el esencial para lograr los rendimientos alcanzados y las diferentes componentes la red TR, la red TT o las capas completamente conectadas no parecen lograr un diferencial entre sí en su capacidad de clasificar esta base de datos particular. A pesar de no poder lograr un diferencial entre las redes, una parte del objetivo fue conseguido al poder conseguir un mejor rendimiento al obtenido utilizando solamente las redes tensoriales como fue planteado originalmente, aunque los resultados conseguidos están lejos del estado del arte para este conjunto de datos (ver Tabla 5).

5.5.4. IMDB

Se realizaron varios experimentos sobre la base *CIFAR-10*, con cinco tipos de redes diferentes, dos redes TT y TR, que a diferencia del resto de los experimentos, reemplazando los *Features Maps* con una capa de *Embedding*, que se denominaron *TT-Text* y *TR-Text* respectivamente. Para estas

dos se probaron diferentes configuraciones variando la cantidad de divisiones τ , el valor de *dropout* luego de la capa de *Embedding*, con valores de m igual a 25 o a 50. Para todos los experimentos el tamaño del *Embedding* se mantuvo constante en 50, además se fijó para que todas las críticas del set tengan en total 200 *tokens*, rellenando las críticas más cortas y truncando las más largas. Las otras tres redes tienen al principio un componente convolucional. Se denominó *CNN-Text*, a la red con la capa completamente conectada encargada de realizar la tarea de clasificación, mientras que se denominó *CNN-TT-Text* y *CNN-TR-Text* a las que tienen las redes TT y TR respectivamente como último componente. Para la red *CNN-Text* se realizaron experimentos variando el valor de *Dropout* entre 0.1 y 0.5, con un incremento fijo igual a 0.1. Mientras que para los modelos *CNN-TT-Text* y *CNN-TR-Text* se experimentó no sólo variando el valor de *Dropout*, también se hicieron ensayos modificando la cantidad de divisiones, como se mencionó en la sección 5.1, la salida de la capa convolucional con los tres diferentes n-gramas, con el tamaño de los filtros convolucionales igual a 100, y luego de dichas convoluciones se realiza una operación de *pooling* de una dimensión sobre cada una de las salidas, dadas todas estas condiciones la salida concatenada es de $\mathbb{R}^{3 \times 100}$, se probó partir la salida en su segunda dimensión en una, dos o cuatro partes, siendo los ensayos con cuatro divisiones los que arrojan mejores resultados, con valor de $\tau = 12$ y $m = 25$.

Red	Precisión	Embedding	m	R	Cantidad de parámetros	τ	Dropout
<i>TT-Text</i>	65.1 %	50	50	50	1406100	5	0.5
<i>TR-Text</i>	63.46 %	50	50	50	1402600	5	0.35
<i>CNN-TT-Text</i>	81.77 %	50	25	50	1375400	12	0.1
<i>CNN-TR-Text</i>	81.65 %	50	25	50	1377900	12	0.2
<i>CNN-Text</i>	82.91 %	50	-	-	1310701	-	0.1

Tabla 11: Mejor Rendimiento de las diferentes redes para la base de datos *IMDB*

Esta base de datos es completamente distinta a las tratadas anteriormente, las tres series de experimentos previamente descriptos, realizaban en esencia tareas similares entre sí, la de clasificar un conjunto de imágenes en un conjunto de categorías. Aunque éstos eran diferentes entre sí y con distintos niveles de complejidad, inclusive con diferencias en la cantidad de dimensiones que tenían, se puede argumentar que las tres eran fundamentalmente semejantes en cuanto a que el tipo de entrada era una imagen. Esto permite afirmar que la naturaleza de estos problemas no presentaba un desafío diferente para las redes diseñadas para realizar estos ensayos. Por esto fue muy valioso agregar esta base de datos que presenta un desafío distinto, es como se indicó en la sección 5.1, un problema de *NLP* (*Natural Language Processing* o Procesamiento de Lenguaje Natural), específicamente de clasificación de sentimiento (positivo o negativo) de las críticas de cine en el conjunto de datos. Para abordar esta nueva problemática como se mencionó anteriormente, se comenzó probando con una red *Vanilla RNN*, que arrojó resultados poco alentadores, ésta alcanzaba alrededor de 50 % de precisión, la misma que se podría conseguir lanzando una manera aleatoriamente. Lógicamente con este tipo de rendimiento, no se puede considerar que la red este funcionando satisfactoriamente, además que no es posible sacar ninguna conclusión interesante para este trabajo.

Utilizando la red TT y la red TR, se lograba en el inicio rendimientos similares a la red *Vanilla RNN*, pero se observaban en los experimentos resultados marginalmente mejores en los conjuntos de

entrenamiento que con el conjunto de validación utilizado. Esto resultó en la experimentación con la técnica de *Dropout* para la salida de las capas de *Embedding*, recordando que en este caso, no se utilizaron los *Features Maps* (Ec. 4.23), alimentando directamente a las redes con el producto de las capas de *Embedding*. Esta técnica permitió corregir el fenómeno de *Overfitting* que se había observado, se experimentaron con varias configuraciones para ambas arquitecturas. En los ensayos realizados para las dos redes se efectuaron diferentes pruebas variando concurrentemente, los valores de inicialización de los tensores, la tasa de *Dropout*, la cantidad de divisiones τ y el rango R , manteniendo constante el valor de m , que en este caso es igual al tamaño de la capa de *Embedding*. Finalmente los mejores resultados se obtuvieron con rango $R = 50$ y una cantidad de visiones igual a 5, con diferentes valores de *Dropout* para cada una de las arquitecturas, 0,5 para la red TT y 0,35 para la red T, Rcon resultados más satisfactorios, obteniendo un rendimiento de 65,1 % y 63,46 % respectivamente.

Con estos resultados se puede aseverar que ambas redes cumplen de manera competente la tarea de catalogar como positiva o negativa una de las críticas de películas que se encuentran en la base de datos. Comparando ambas redes entres sí (ver Tabla 11), con una cantidad similar de parámetros, se puede distinguir que la red TT funciona mejor que la red TR, aunque el promedio de los experimentos se encuentran en niveles bastante similares, el modelo basado en la descomposición *Tensor Train* es levemente superior en su capacidad de clasificación.

Con el mismo objetivo que en el caso de los ensayos realizados con la base *CIFAR-10*, se planteó un modelo diferente basado en una arquitectura que puede ser considerada como novedosa, para mejorar los resultados obtenidos para las redes tensoriales detalladas arriba. Como se mencionó en la sección 5.1, se utilizó como base el trabajo desarrollado por Kim 2014 [55] para diseñar una red con tres filtros convoluciones que funcionan como para extraer los *features* para un agrupamiento de n-gramas. La idea detrás de esto, es que los n-gramas de tamaños 3, 4 y 5 detecten alguna secuencia de palabras que sea un fuerte indicador de los sentimientos de esta crítica, con una operación de *pooling* con el objetivo de detectar los valores más importantes entre cada uno de los *features* extraídos por los n-gramas.

Como se mencionó en la sección de los experimentos, se realizaron varios ensayos para las redes con un componente convolucional variando el valor de *Dropout*, esto indica la probabilidad de que un cierto valor de la salida de la capa convolucional se vuelva 0 al azar. El objetivo de esta técnica es que las redes encargadas de clasificar en base a los *features* elaborados por el componente convolucional, aprendan a clasificar sin un sesgo específico a un valor en particular. Luego de realizar varios experimentos se encontró una mejora al aplicar esta técnica, implicando un valor distinto de cero, y se notó a medida que se aumenta el valor en este caso, una caída en la eficiencia a la hora de clasificar cuando se llegan a un *Dropout* igual o mayor a 0.5. Entre esos dos valores se eligieron como resultados los experimentos con mejores rendimientos pero no hay una diferencia importante entre esos resultados. Además como se mencionó en la sección de experimentos, también se ensayó con diferente cantidad de divisiones, en las distintas configuraciones ensayadas, finalmente se eligió un valor de τ igual a 12. Esto remarca la importancia de la experimentación con diferentes configuraciones para encontrar los mejores resultados modificando los hiperparámetros de manera discreta.

A la hora de analizar los resultados obtenidos para estos experimentos con esta variante para las diferentes redes, todas consiguieron resultados similares para el conjunto de datos elegidos, siendo mejor el resultado para el modelo denominado *CNN-Text* con una precisión de 82,91 %. Mientras que las redes *CNN-TT-Text* y *CNN-TR-Text* obtuvieron un rendimiento de 81,77 % y 81,65 % respectivamente en la tarea de clasificar una crítica con sentimiento positivo o negativo. Estos resultados son de un orden similar pero se puede aseverar que la red *CNN-Text* obtuvo los mejores resultados. Esto marca una diferencia con los experimentos análogos realizados para la

base *CIFAR-10*, dado que con un esquema similar, con un componente convolucional generando los *features* para las arquitecturas TT y TR, en el caso anterior no se pudo notar una desigualdad notable entre éstas y la red con la capa completamente conectada actuando como clasificador. En el caso de esta base de datos las redes tensoriales no obtuvieron ninguna ventaja comparativa contra la red completamente conectada que es la encargada de clasificar en la red de mejor rendimiento. Éstas aún así demostraron ser claramente superiores a la redes tensoriales, sin el componente convolucional, consiguiendo un resultado satisfactorio que como en los casos anteriores no alcanzan al actual estado del arte (ver Tabla 5).

Al igual que en el resto de los experimentos se puede afirmar que en todos los casos se consiguió un nivel que permite afirmar que estas redes cumplen con su tarea de manera satisfactoria. Aún más, se logró esto con una base de datos complementamente diferente al anterior probando con técnicas de preprocesamiento completamente distintas y haciendo ensayos con diferentes arquitecturas novedosas.

CAPÍTULO 6: CONCLUSIONES

En este capítulo se encuentran las conclusiones más relevantes que se encontraron como fruto de este trabajo. Empezando por los puntos de mayor interés relacionados a la investigación realizada, resaltando los ensayos más fundamentales y las deducciones que se extraen de ellos. Luego se desarrollan las conclusiones extraídas de los experimentos, teniendo en cuenta el marco teórico elaborado y toda la experiencia ganada en el desarrollo técnico.

Finalmente, se exponen las que se consideran las líneas de investigación más interesantes que surgen de todas las conclusiones previamente elaboradas, proponiendo variaciones a la propuesta exhibida. Ofreciendo variaciones en las bases de datos utilizadas, como también modificaciones en los hiperparámetros. Además de variantes en las arquitecturas propuestas y algunas técnicas que no fueron usadas en este trabajo pero serían de gran provecho poder aplicarlas en este tipo de investigación.

6.1. Conclusiones Generales

En los últimos años se ha popularizado el uso de herramientas de *Deep Learning* para una gran variedad de problemas complejos con buenos resultados. El crecimiento en el volumen de datos involucrados y el aumento en el tamaño y la complejidad de las redes neuronales utilizadas lleva a buscar soluciones para tratar contra este nuevo problema. Así surge la idea de poder reducir su tamaño o modificar su arquitectura para intentar mejorar su desempeño y bajar sus requerimientos sin comprometer de manera severa su funcionamiento. En los trabajos citados en esta tesis que refieren a la compresibilidad de las redes se aplican diferentes enfoques, todos tienen como idea principal comprimir capas o inclusive redes completas utilizando técnicas de descomposición tensorial. Este enfoque muestra muy buenos resultados en algunos trabajos [28, 63, 74]. La idea de comprimir redes y los beneficios que trae en tiempos y en el almacenamiento hacen que sea una línea de investigación muy interesante.

Por otro lado, el enfoque utilizado en los trabajos relacionados con el concepto de expresividad que plantea Cohen *et al.* 2016 [18], continuado a su vez en trabajos sucesivos [20, 51, 52], utilizan a las descomposiciones tensoriales no sólo como una herramienta de compresión sino que tienen una correspondencia con una arquitectura de una red de *Deep Learning*. También relacionan los hiperparámetros (rangos) de éstas con su poder expresivo permitiendo dar un marco teórico detrás del funcionamiento de las redes y poder entender los fundamentos sobre los cuales éstas se basan. Teniendo en cuenta esto lo vuelve muy interesante como un concepto sobre el cual desarrollar y fue uno de los puntapiés de este trabajo. Sin embargo, es claro que el avance sobre este todavía está en una primera etapa de investigación y su aplicación práctica todavía se encuentra lejos de ser de uso popular.

6.2. Conclusiones Experimentales

A pesar de que las diferencias no son grandes, se puede enunciar generalmente que la arquitectura TT funcionó mejor que la arquitectura TR propuesta. En redes con la misma cantidad de parámetros mostró tener una mejor productividad a la hora de clasificar diferentes bases de datos. Siendo *CIFAR-10* (ver Tabla 10) el único caso donde esta arquitectura demostró un mejor rendimiento, aunque de manera casi marginal. De la misma manera es importante remarcar que tanto para el caso previamente citado, como para el caso de las redes *CNN-TT-Text* y *CNN-TR-Text* para la base *IMDB* (ver Tabla 11) y para el ensayo para *MNIST* (ver Tabla 6) con las redes *TT* y *TR*, las diferencias en los mejores resultados son pocos significativas. Mientras que para el caso de la base datos *Fashion-MNIST* (ver Tabla 8) para las redes *TT* y *TR*.

Otro resultado interesante es el pobre rendimiento de las arquitecturas *paralela* y *serializada*. Los resultados preliminares de las pruebas con estos no fueron buenos para los sets de datos menos complejos, *MNIST* y *Fashion-MNIST*, e inferior a la arquitectura *compartida*, que fue utilizada para los resultados mencionados previamente. La cantidad de parámetros de estas arquitecturas es uno de los factores que explican este rendimiento. Esto junto a la disposición de los tensores en estas redes vuelven dificultoso el aprendizaje, estas dependiendo de la cantidad de tensores pueden experimentar el ya mencionado *Vanishing gradient problem*.

Mientras que para las redes *TT-Text* y *TR-Text* para los ensayos con *IMDB* (ver Tabla 11) los mejores rendimientos para la arquitectura TT es más apreciable. Esto puede deberse a que este es un set más complejo la red con la descomposición TR necesita ser más ancha, esto en términos de la red significaría un R más grande. Esta presunción se basa en la experiencia con los otras bases, en los casos de *MNIST* y *Fashion-MNIST*, cuando a una red con un m lo suficientemente significativo se le incrementa el R lo suficiente el rendimiento de las redes correspondientes a ambas descomposiciones tienden a igualarse.

En cuanto al rendimiento respecto de los hiperparámetros de las redes, se puede afirmar que consistentemente en todos los experimentos con las redes *TT* y *TR*, tanto para la base *MNIST* y para *Fashion-MNIST*, que a medida que aumenta el tamaño de los *features maps* y por lo tanto el tamaño de la entrada de las redes, m , el desempeño mejora de manera considerable. Desde un punto de vista teórico, el valor de m representa la magnitud de las dimensiones, tanto del *feature tensor* $\Phi(\mathbf{X})$, el producto de todos los *Features maps* por cada fragmento de las entradas, como del *weight tensor* \mathcal{W}^y de pesos entrenables ambos de τ dimensiones, el último es el objeto de las correspondientes descomposiciones. Por razones de conveniencia este tensor es de forma de un cubo n -dimensional, mientras más grande el valor de m , aún mayor es este tensor teórico representado por la descomposición y más pesos entrenables tiene la descomposición en sí. Además se eleva la extensión del *tensor feature*, y por lo tanto la cantidad de *features* que representan a cada entrada, por todas estas razones es lógico que aumentar m implique una mejora en el rendimiento de las respectivas redes.

Lo mismo que se afirmó para m se puede afirmar de una manera similar para el valor de Rango R , el aumento de éste está asociado a un incremento en el rendimiento de la red, tanto para la base *MNIST* y para *Fashion-MNIST*, con los experimentos con las redes *TT* y *TR*. El rango, en el contexto de la descomposición representa los rangos auxiliares de cada tensor de la descomposición, como fue explicado anteriormente para lograr replicar la estructura de una red *RNN*, éste es compartido a través de toda la red, en el caso de la red neuronal este representa en términos coloquiales el ancho de la misma, en esta línea una red más ancha, tiene más parámetros, puede aprender más y por lo tanto mejorar su eficacia. Aún así como fue notado en las conclusiones previas, éstas tienen un límite, por más anchas que se vuelvan, parecen tener una cota en cuanto a su capacidad para clasificar exitosamente, esto fue consistentemente para ambas redes en las distintas bases de datos en los experimentos realizados. El poder expresivo de la red está directamente relacionado al ancho y a las dimensiones del tensor teórico representado por la descomposición, y es por lo tanto lógico asumir que el aumento de las variables asociadas a estos, m y R respectivamente, produzcan un aumento en el rendimiento. Debido a esto, se puede asegurar que el crecimiento de éstas le da a la red más poder expresivo y éste puede realizar su trabajo de manera más eficiente.

Se estableció que para ambos valores m y R , tienen una correlación positiva entre su valor nominal y el rendimiento de las redes, pero es importante notar su impacto en otras variables de la red. En las respectivas ecuaciones para calcular la cantidad de parámetros para la red *Tensor Train* (Ec. 4.56) y para la red *Tensor Ring* (Ec. 4.55), la relación de los hiperparámetros es diferente, para ambas redes m tiene una correlación lineal, para el caso del rango es cuadrática para ambos casos, esto implica que aumentar el valor de R es decididamente más costo que un incremento en m para la cantidad de parámetros de la red. Se ve explícitamente el comportamiento descrito en la Figura 57, que grafica la cantidad de parámetros en función del rango de redes con diferentes valores de m . Esto se replica de manera similar para la cantidad de operaciones para las dos redes, en este aspecto para la red *TT*, la ecuación (Ec. 4.47) que representa la cantidad de operaciones para realizar la acción de *feedforward* para un *batch* de entrada, la correlación con m es lineal, mientras que la misma para R la correlación es cuadrática, de manera análoga con la cantidad de parámetros. Entretanto el orden de las operaciones para la red *TR* (Ec. 4.50) se comporta de una manera distinta al de la otra red, para la misma la relación con m es una vez más lineal pero la relación con R es cúbica debido a la operación de traza presente en esta arquitectura. Al igual que en el caso de la cantidad de parámetros, para el orden de operaciones todavía resulta más punitivo el aumento de m respecto de un incremento en el rango, especialmente para el caso de la red *TR*, dada su relación de tercer grado, lo vuelve especialmente caro. Teniendo en cuenta esto, es pertinente afirmar que es imperante a la hora de establecer los hiperparámetros elegir un valor de R que permita alcanzar buenos niveles de funcionamiento a la red, sopesando el impacto de este cambio en su funcionamiento y su desempeño.

Siguiendo esta línea de conclusiones, cuando se compara la red TT y la red TR en tiempo y cantidad de operaciones emerge la que es probablemente la mayor diferencia entre éstas, el tiempo devenido de la cantidad de operaciones. Se refirió previamente a la disparidad entre la cantidad de parámetros, no obstante se encontraron en diferentes bases de datos rendimientos similares para cantidad de parámetros casi equivalentes (ver Tablas 6, 8 y 10), pero detrás de esto es importante notar la desigualdad entre los tiempos que impacta, no sólo en los de entrenamientos, principalmente en la etapa de *backpropagation*, sino en el funcionamiento del proceso de *feedforward*, que es un factor importante a evaluar para poner modelos a realizar tareas productivas. Es explícito en los órdenes calculados para la red *Tensor Train* (Ec. 4.47) y para la red *Tensor Ring* (Ec. 4.50), la diferencia se origina principalmente en los grados del rango de ambas, ya mencionado en el párrafo anterior, el orden en el caso de la red TR que es casi cúbico, con $R \approx R'$, la vuelve claramente más costosa que el caso de la red TT, donde el orden para el rango es cuadrática. La operación de traza es otro factor interesante a analizar, la dimensión del rango $R_{\tau+2}$ en los estados ocultos y en el último tensor es sobre la cual se ejecuta. Se podría interpretar como que cada una de estas dimensiones es una descomposición TT y a través de la traza la descomposición TR resulta la suma de todas esas descomposiciones, para todos los ensayos de este trabajo se estableció $R_{\tau+2} = 10$ (ver Figura 27). Otra de las conclusiones a las que se llegó a través de la experimentación es que la inicialización de los parámetros es clave a la hora de lograr buenos resultados en los experimentos, fue un gran avance dentro del proyecto encontrar la inicialización y los parámetros correctos para este fin.

Teniendo en cuenta las conclusiones previamente desarrolladas, se puede afirmar con confianza que el objetivo del trabajo de poder desarrollar este nuevo tipo de arquitectura, e implementarla se logró con éxito. Pudiendo realizar experimentos con diferentes bases de datos, distintos en su naturaleza y complejidad, y con diferentes diseños en la etapa de preprocesamiento. Los experimentos demostraron en todos los casos tener resultados satisfactorios, pudiendo demostrar el funcionamiento de la arquitectura propuesta. En cuanto a su comparación con la red *Tensor Train* no se pudieron conseguir rendimientos superiores en los ensayos planteados, variando los distintos hiperparámetros a través de los diferentes bases utilizados. En muchos casos se pudo conseguir rendimientos iguales e incluso marginalmente mejores, pero no se puede proclamar a la red *Tensor Ring* como una alternativa superadora. A pesar de esto se pudo establecer la viabilidad de esta red, en la siguiente sección 6.3, se plantean distintas líneas de investigación que se proponen en base a lo desarrollado en este trabajo.

6.3. Futuras líneas de investigación

Este trabajo intenta contribuir con el estado del arte de la investigación, continuando con una de las varias líneas de investigación sobre la relación de las descomposiciones tensoriales con las redes de aprendizaje profundo, y la aplicación de éstas a este campo. Este desarrollo abre el abanico para seguir profundizando especialmente con este tipo de descomposiciones, tratando de llevar el valor agregado de la teoría a una aplicación en este área sobre la cual se realizan descubrimientos continuamente y se utiliza cada vez más en un universo de aplicaciones diferentes.

Una de las alternativas que surgen a partir de los ensayos realizados utilizando las arquitecturas propuestas es probar diferentes configuraciones de *features maps*. Una variante que quedo por explorar, es probar con valores de m mayores, en los experimentos realizados con la base *MNIST* y con *Fashion-MNIST*, se probaron con valores fijos de m , aumentando el valor del rango, probando con hacer la red más ancha, encontrando un límite práctico al aumento de rendimiento que fue discutido en la sección anterior. El concepto teórico de aumentar el valor de m , además de aumentar la cantidad de *features* para poder alimentar a la red, es el de volver los tensores teóricos que forman la ecuación que tiene como salida la *score function* (Ec. 4.6), tanto el *tensor feature* como el *weight*

tensor más grandes. Siguiendo el mismo razonamiento de lo que sucedió con el valor de R y teniendo en cuenta también el tamaño de la entrada para estas bases, se alcanzará también un límite en la cantidad de *features* que le aportarán a la red una mejora en su capacidad de clasificación. Pero esto no fue explorado para este trabajo con esta implementación, además como se explicó extensivamente en la sección previa, el aumento en el rango tiene un impacto sustantivo en la cantidad de parámetros y especialmente en la cantidad de operaciones para la acción de *feedforward*, el efecto que tiene el incremento de m es mucho menor para ambas variables de la red. Esto implica que no sólo podría generar un impacto positivo en el rendimiento final de la red, también es interesante analizar el balance entre valores de m y R en este caso, de alguna manera intercambiando los papeles con respecto al análisis realizado, pudiendo generar una mejor clasificación utilizando la misma cantidad de parámetros con menor cantidad de operaciones.

En esta misma línea, para mejorar los resultados de ambas redes para las bases *CIFAR-10* y para *IMDB*, se probaron diferentes diseños y algoritmos para generar los *features* de entrada para las redes. Probando por ejemplo con un componente convolucional para construir éstos para *CIFAR-10* (ver Figura 43), utilizando *embeddings* para *IMDB*, también combinándolo con un componente convolucional (ver Figura 46) todo con el objetivo de mejorar los resultados encontrados para los mismos datos para el *Feature Map* (Ec. 4.23) definido en un principio. La construcción de **features** es un área muy interesante, con muchas posibilidades de experimentación con las bases utilizadas. Una de las variantes para continuar con esta línea sería probar con otros tipos de *Features Maps*, estudiando la combinación entre éstos y las redes, y si la mezcla de éstos provee una mejora sustantiva. Buscando si hay alguna configuración específica u otro tipo que dé resultados superiores para estas redes, e investigar el porqué, tal vez arrojando más luz sobre la naturaleza y las características de estas redes.

De la misma manera en la cual hay mucho potencial usando diferentes *Features Maps* para estas redes, también se abren muchas líneas de investigación para continuar experimentando con este tipo de redes con otras bases de datos. En este trabajo se exploraron diferentes tipos de datos, se hicieron ensayos con cuatro conjuntos diferentes, con distintos niveles de complejidad y de diferente naturaleza entre sí. De éstos, tres se pueden definir como problemas de clasificación de imágenes, teniendo imágenes en escalas de grises más simples (*MNIST*), imágenes del mismo tipo aunque más complejas de clasificar (*Fashion-MNIST*), e imágenes en colores (*CIFAR-10*). Mientras que el restante es un problema *NLP* de clasificación de sentimientos, todos estos conjuntos permitieron realizar varios ensayos con diferentes configuraciones y sacar conclusiones sobre la arquitectura propuesta. Los experimentos elaborados permitieron aseverar el funcionamiento de ambas redes, pero se abren muchas posibilidades con diferentes tipos de problemas para explorar especialmente por la naturaleza de la red. En este trabajo se eligió utilizar la base *IMDB* para tener un testigo como problema en el cual una red RNN puede ser elegida como una opción predilecta para resolverlo, esto abre el abanico a una gran variedad de problemas en los cuales este tipo de redes son muy aptos para solucionar de manera eficiente. Específicamente el problema abordado se puede generalizar como un problema de predicción secuencial, que se podría resolver con una RNN *muchos-a-uno*, siendo una secuencia de muchos períodos, el texto de la crítica, con una salida, la clasificación del sentimiento. En esta misma línea hay otras bases que se usan como el punto de referencia para tareas de la misma naturaleza como el etiquetado de noticias, un ejemplo de ésta podría ser *AG News*[72] un conjunto de noticias que debe ser etiquetado en una de cuatro categorías. También se podrían realizar ensayos con otros tipos de textos aptos para la clasificación de sentimientos como *YELP*, *Amazon* o *SST* estos son otras bases del mismo tipo utilizadas frecuentemente, que permiten evaluar el rendimiento y compararlo con el encontrado en este trabajo, permitiendo trazar paralelismos variando las configuraciones de la red. Otro tipo de tareas para las cuales podría utilizarse la arquitectura propuesta, es para la clasificación de tópicos, donde el conjunto de datos más utilizado

es *DBpedia*, basado en un grupo de artículos de *Wikipedia*, o para la inferencia del lenguaje natural con el set *SNLI*, para clasificar oraciones que actúan como hipótesis que deben determinarse como verdaderas, falsas o neutras, éste es uno de los problemas más complejos a resolver en el caso de NLP. Estos problemas podrían ser resueltos por la arquitectura como esta planteada en este trabajo y serían interesantes para evaluar el funcionamiento de la red.

Una variación que quedó por estudiar en este trabajo y sería interesante de profundizar avanzando en base a lo realizado, es probar en diferentes valores de Rangos para la red *Tensor Ring*. Como se puede ver en la Figura 27 para la red TR, el caso de tener el núcleo compartido obliga a que todos los rangos compartido $R_1 = R_2 = \dots = R_{\tau+1}$ tengan el mismo valor, pero eso no implica que $R_{\tau+2}$ sea igual al resto de los rangos, para el caso de TT éste es igual a uno y para este trabajo se decidió que este último rango tenga el mismo valor que el resto. Como se mencionó en la sección de Implementación 5.1, se fijó $R_{\tau+2} = 10$, que es el rango sobre el cual se hace la operación de traza, con un valor de $R_{\tau+2} = R$ en los experimentos anteriores, como se aludió previamente la cantidad de operaciones para la acción de *feedforward* a medida que R aumenta se vuelve muy complejo, por eso se decidió darle un valor fijo menor. Esto abre un abanico para probar con diferentes valores de $R_{\tau+2}$, para encontrar cual es el valor que resulta en el mejor rendimiento y poder darle una interpretación a los resultados encontrados. La teoría es que debería tener una relación con R , pero no necesariamente debe ser equivalente en valor, basados en la experiencia de este trabajo.

Durante el desarrollo de este trabajo salió un nuevo desarrollo [51] que pertenece a los mismos autores de varios trabajos sobre los que está basado éste, plantea el mismo concepto pero desarrollado para modelos tensoriales generalizados. Esto implica ampliar la teoría desarrollada para incluir varios productos no lineales al producto exterior entre los *Feature maps* de las entradas y los estados ocultos de la red, esto permite que las redes se asimilen aún más la teoría desarrollada con los modelos que se utilizan en ambientes productivos en la actualidad. Esta teoría podría incluirse a la arquitectura propuesta en este trabajo. Agregando diferentes productos mezclados con funciones de activación no lineales. Esto en teoría permitiría mejorar las dependencias de largo plazo, y tendría como consecuencia una mejora en el funcionamiento general de la red. Otra idea que no fue explorada en este trabajo, pero puede también apuntar a resolver las dependencias de largo plazo, especialmente el problema del *Exploding gradient* que fue aludido ya, es el aplicar una técnica conocida como *gradient clipping* o recorte de gradiente, que sirve para facilitar la acción de *backpropagation* para redes donde en las continuas operaciones de *feedforward* el gradiente se vuelve muy elevado, por lo que éste es recortado para poder ajustarlo de manera más lenta pero efectiva. Estas propuestas tienen en común abordar el problema que presenta la red a la hora de aprender con un valor de τ muy grande, e incurre en el problema del *Exploding/Vanishing gradient*. Otra manera de abordar éste y seguir en el camino de alinear la teoría de estos trabajos con el estado de arte actual de las redes, es el de desarrollar ésta sobre redes RNN más complejas. Como se explicó en la sección A.3, una red GRU (ver Figura 59) aborda este problema efectivamente teniendo una compuerta (*update gate*) encargada de cuánto se debe tener en cuenta la información de los estados ocultos y otra (*reset gate*) encargada de cuánto debe olvidar de los estados anteriores. Este diseño es especialmente práctico para las deficiencias presentadas por la arquitectura en este trabajo. De la misma manera en la sección A.2 se describen las redes LSTM (ver Figura 58), que tienen un mecanismo parecido pero tienen tres compuertas, a diferencia de la GRU en vez de *reset gate* tiene dos compuertas una específicamente diseñada para decidir que olvidar del estado anterior (*forget gate*) y otra que define el próximo estado oculto (*output gate*), esta red es más pesada en cuanto a la cantidad de parámetros y procesamiento respecto de la red GRU, pero demuestra ser consistentemente mejor para problemas más complejos, especialmente con mayor cantidad de pasos. El próximo paso sería intentar implementar una arquitectura tensorial con alguna o ambas de estas redes, inclusive utilizando más de una descomposición para llevarla a cabo. En este sentido es muy

importante el trabajo mencionado previamente que generaliza la teoría para diferentes tipos de productos y agrega componentes no lineales, y puede utilizarse como una piedra fundamental para el desarrollo teórico y práctico de las redes, debido a la necesidad de estas operaciones para implementar las ecuaciones para una red LSTM (Ec. A.6)(Ec. A.4)(Ec. A.8)(Ec. A.5)(Ec. A.7) como para una red GRU (Ec. A.11)(Ec. A.11)(Ec. A.9).

Por otro lado, una alternativa a utilizar redes recurrentes más complejas es intentar ir por el camino de la compresión, utilizando trabajos referenciados, como el de Novikov *et al.* 2015 [74], profundizado por el mismo equipo más adelante [28]. La posibilidad abierta es realizar algo análogo a lo propuesto por este trabajo. En los ensayos citados se utiliza la descomposición TT para poder comprimir tanto capas complementamente conectadas, como capas convolucionales. El objetivo detrás de esto, es el mismo que el de este trabajo pero por otros medios, poder lograr buenos rendimientos, intentando reducir la cantidad de parámetros y operaciones. De la misma manera se podría utilizar una descomposición TR para intentar obtener resultados similares, teniendo en cuenta que tiene las mismas propiedades y ventajas que la descomposición TT, es un algoritmo más robusto y es inmune a la maldición de la dimensionalidad en comparación con otras descomposiciones como la CPD, recordando que la descomposición TR es una generalización de la TT. Otro desarrollo publicado durante la elaboración de este trabajo [50] propone descomponer las capas de *Embedding* usando una descomposición TT, alineado con el uso de esta técnica en los experimentos también sería consistente con las líneas abiertas experimentar con esta variante y explorar su funcionamiento, intentando de alguna manera “tensorizar” toda la red.

REFERENCIAS

- [1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [2] Richard E Bellman. *Adaptive control processes: a guided tour*, volume 2045. Princeton university press, 2015.
- [3] Yoshua Bengio, Paolo Frasconi, and Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *IEEE international conference on neural networks*, pages 1183–1188. IEEE, 1993.
- [4] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.
- [5] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Praseoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.
- [6] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing*, pages 227–236. Springer, 1990.
- [7] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.
- [8] Xingwei Cao, Xuyang Zhao, and Qibin Zhao. Tensorizing generative adversarial nets. In *2018 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pages 206–212. IEEE, 2018.
- [9] J Douglas Carroll and Jih-Jie Chang. Analysis of individual differences in multidimensional scaling via an n-way generalization of “eckart-young” decomposition. *Psychometrika*, 35(3):283–319, 1970.
- [10] Raymond B Cattell. The three basic factor-analytic research designs—their interrelations and derivatives. *Psychological bulletin*, 49(5):499, 1952.
- [11] Augustin Cauchy. Méthode générale pour la résolution des systemes d’équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.
- [12] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.
- [13] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [14] Andrzej Cichocki, Namgil Lee, Ivan Oseledets, Anh-Huy Phan, Qibin Zhao, Danilo P Mandic, et al. Tensor networks for dimensionality reduction and large-scale optimization: Part 1 low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4-5):249–429, 2016.
- [15] Andrzej Cichocki, Danilo Mandic, Lieven De Lathauwer, Guoxu Zhou, Qibin Zhao, Cesar Caiafa, and Huy Anh Phan. Tensor decompositions for signal processing applications: From two-way to multiway component analysis. *IEEE Signal Processing Magazine*, 32(2):145–163, 2015.

- [16] Andrzej Cichocki, Anh-Huy Phan, Qibin Zhao, Namgil Lee, Ivan Oseledets, Masashi Sugiyama, Danilo P Mandic, et al. Tensor networks for dimensionality reduction and large-scale optimization: Part 2 applications and future perspectives. *Foundations and Trends® in Machine Learning*, 9(6):431–673, 2017.
- [17] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*, 2012.
- [18] Nadav Cohen, Or Sharir, and Amnon Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.
- [19] Nadav Cohen and Amnon Shashua. Simnets: A generalization of convolutional networks. *arXiv preprint arXiv:1410.0781*, 2014.
- [20] Nadav Cohen and Amnon Shashua. Convolutional rectifier networks as generalized tensor decompositions. In *International Conference on Machine Learning*, pages 955–963, 2016.
- [21] Misha Denil, Babak Shakibi, Laurent Dinh, Nando De Freitas, et al. Predicting parameters in deep learning. In *Advances in neural information processing systems*, pages 2148–2156, 2013.
- [22] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.
- [23] Phoebe MR DeVries, Fernanda Viégas, Martin Wattenberg, and Brendan J Meade. Deep learning of aftershock patterns following large earthquakes. *Nature*, 560(7720):632, 2018.
- [24] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.
- [25] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.
- [26] Rasool Fakoor, Faisal Ladhak, Azade Nazi, and Manfred Huber. Using deep learning to enhance cancer diagnosis and classification. In *Proceedings of the international conference on machine learning*, volume 28. ACM New York, USA, 2013.
- [27] Kuniyiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.
- [28] Timur Garipov, Dmitry Podoprikin, Alexander Novikov, and Dmitry Vetrov. Ultimate tensorization: compressing convolutional and fc layers alike. *arXiv preprint arXiv:1611.03214*, 2016.
- [29] Felix A. Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural Computation*, 12:2451–2471, 1999.
- [30] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.
- [31] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

- [32] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.
- [33] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.
- [34] Alex Graves. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*, pages 5–13. Springer, 2012.
- [35] Wolfgang Hackbusch and Stefan Kuhn. A new scheme for the tensor representation. *Journal of Fourier analysis and applications*, 15(5):706–722, 2009.
- [36] R. A. Harshman. Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multi-modal factor analysis. *UCLA Working Papers in Phonetics*, 16:1–84, 1970.
- [37] Johan Håstad. Tensor rank is np-complete. *Journal of Algorithms*, 11(4):644–654, 1990.
- [38] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.
- [39] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.
- [40] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.
- [41] Frank L Hitchcock. Multiple invariants and generalized rank of a p-way matrix or tensor. *Journal of Mathematics and Physics*, 7(1-4):39–79, 1928.
- [42] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.
- [43] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.
- [44] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [45] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.
- [46] Yanping Huang, Yonglong Cheng, Dehao Chen, HyounJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018.
- [47] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- [48] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.
- [49] Michael I Jordan. Serial order: A parallel distributed processing approach. In *Advances in psychology*, volume 121, pages 471–495. Elsevier, 1997.

- [50] Valentin Khrulkov, Oleksii Hrinchuk, Leyla Mirvakhabova, and Ivan Oseledets. Tensorized embedding layers for efficient model compression. *arXiv preprint arXiv:1901.10787*, 2019.
- [51] Valentin Khrulkov, Oleksii Hrinchuk, and Ivan Oseledets. Generalized tensor models for recurrent neural networks. *arXiv preprint arXiv:1901.10801*, 2019.
- [52] Valentin Khrulkov, Alexander Novikov, and Ivan Oseledets. Expressive power of recurrent neural networks. *arXiv preprint arXiv:1711.00811*, 2017.
- [53] Henk AL Kiers. Towards a standardized notation and terminology in multiway analysis. *Journal of Chemometrics: A Journal of the Chemometrics Society*, 14(3):105–122, 2000.
- [54] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.
- [55] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.
- [56] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [57] Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.
- [58] Jean Kossaifi, Zachary Lipton, Aran Khanna, Tommaso Furlanello, and Anima Anandkumar. Tensor regression networks. *arXiv*, 2017.
- [59] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www.cs.toronto.edu/kriz/cifar.html*, 55, 2014.
- [60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [61] Joseph B Kruskal. Three-way arrays: rank and uniqueness of trilinear decompositions, with application to arithmetic complexity and statistics. *Linear algebra and its applications*, 18(2):95–138, 1977.
- [62] Quoc V Le, Navdeep Jaitly, and Geoffrey E Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.
- [63] Vadim Lebedev, Yaroslav Ganin, Maksim Rakhuba, Ivan Oseledets, and Victor Lempitsky. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.
- [64] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [65] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [66] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. <http://yann.lecun.com/exdb/mnist/>, 2010.

- [67] Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.
- [68] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [69] Minh-Thang Luong, Ilya Sutskever, Quoc V Le, Oriol Vinyals, and Wojciech Zaremba. Addressing the rare word problem in neural machine translation. *arXiv preprint arXiv:1410.8206*, 2014.
- [70] Andrew L Maas, Raymond E Daly, Peter T Pham, Dan Huang, Andrew Y Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pages 142–150. Association for Computational Linguistics, 2011.
- [71] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.
- [72] Shervin Minaee, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. Deep learning based text classification: A comprehensive review. *arXiv preprint arXiv:2004.03705*, 2020.
- [73] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- [74] Alexander Novikov, Dmitrii Podoprikin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450, 2015.
- [75] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.
- [76] Ivan V Oseledets, DV Savostianov, and Eugene E Tyrtshnikov. Tucker dimensionality reduction of three-dimensional arrays in linear time. *SIAM Journal on Matrix Analysis and Applications*, 30(3):939–956, 2008.
- [77] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.
- [78] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS 2017 Workshop on Autodiff*, 2017.
- [79] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [80] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning separable filters. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2754–2761, 2013.
- [81] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

- [82] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.
- [83] Berkant Savas and Lars Eldén. Handwritten digit classification using higher order singular value decomposition. *Pattern recognition*, 40(3):993–1003, 2007.
- [84] Nicholas D Sidiropoulos, Lieven De Lathauwer, Xiao Fu, Kejun Huang, Evangelos E Papalexakis, and Christos Faloutsos. Tensor decomposition for signal processing and machine learning. *IEEE Transactions on Signal Processing*, 65(13):3551–3582, 2017.
- [85] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [86] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.
- [87] Edwin Stoudenmire and David J Schwab. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, pages 4799–4807, 2016.
- [88] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
- [89] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. *arXiv preprint arXiv:1511.06067*, 2015.
- [90] tensornetwork.org. Tensor network. <http://tensornetwork.org/>, 2019. [Online;].
- [91] Tan Thongtan and Tanasanee Phienthrakul. Sentiment classification using document embeddings trained with cosine similarity. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*, pages 407–414, 2019.
- [92] Ledyard R Tucker. Implications of factor analysis of three-way matrices for measurement of change. *Problems in measuring change*, 15:122–137, 1963.
- [93] M Alex O Vasilescu and Demetri Terzopoulos. Multilinear analysis of image ensembles: Tensorfaces. In *European Conference on Computer Vision*, pages 447–460. Springer, 2002.
- [94] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066, 2013.
- [95] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.
- [96] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.
- [97] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3891–3900. JMLR. org, 2017.
- [98] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *arXiv preprint arXiv:1606.05535*, 2016.
- [99] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

APÉNDICE

A.1. Demostraciones y ecuaciones

Prueba ecuación (Ec. 4.18):

$$\begin{aligned}
 \ell^y(\mathbf{X}) &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=1}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \\
 &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=2}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot \langle \mathbf{f}_\theta(\mathbf{x}^{(1)}), \mathbf{g}^{(1)}(r_1, r_2) \rangle \\
 &= \sum_{r_1=1}^{R_1} \sum_{r_2=1}^{R_2} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=2}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot h_{r_1, r_2}^{(1)} \\
 &= \sum_{r_1=1}^{R_1} \sum_{r_3=1}^{R_3} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=3}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot \sum_{r_2=1}^{R_2} \langle \mathbf{f}_\theta(\mathbf{x}^{(2)}), \mathbf{g}^{(2)}(r_2, r_3) \rangle \cdot h_{r_1, r_2}^{(1)} \quad (\text{Ec. A.1}) \\
 &= \sum_{r_1=1}^{R_1} \sum_{r_3=1}^{R_3} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=3}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot h_{r_2, r_3}^{(2)} \\
 &= \dots \\
 &= \sum_{r_1=1}^{R_1} \sum_{r_\tau=1}^{R_\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(\tau)}), \mathbf{g}^{(\tau)}(r_\tau, r_{\tau+1}) \rangle \cdot h_{r_\tau, r_{\tau+1}}^{(\tau-1)} = \sum_{r_1=1}^{R_1} h_{r_1, r_{\tau+1}}^{(\tau)} = Tr(\mathbf{H}^{(\tau)}) \\
 &= h^{(\tau+1)}
 \end{aligned}$$

Prueba ecuación (Ec. 4.33):

$$\begin{aligned}
 \ell^y(\mathbf{X}) &= \sum_{r_2=1}^{R_2} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=1}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \\
 &= \sum_{r_2=1}^{R_2} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=2}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot \langle \mathbf{f}_\theta(\mathbf{x}^{(1)}), \mathbf{g}^{(1)}(r_1, r_2) \rangle \\
 &= \sum_{r_2=1}^{R_2} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=2}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot h_{r_1}^{(1)} \\
 &= \sum_{r_3=1}^{R_3} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=3}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot \sum_{r_2=1}^{R_2} \langle \mathbf{f}_\theta(\mathbf{x}^{(2)}), \mathbf{g}^{(2)}(r_2, r_3) \rangle \cdot h_{r_1, r_2}^{(1)} \quad (\text{Ec. A.2}) \\
 &= \sum_{r_3=1}^{R_3} \dots \sum_{r_\tau=1}^{R_\tau} \prod_{t=3}^{\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(t)}), \mathbf{g}^{(t)}(r_t, r_{t+1}) \rangle \cdot h_{r_2}^{(2)} \\
 &= \dots \\
 &= \sum_{r_\tau=1}^{R_\tau} \langle \mathbf{f}_\theta(\mathbf{x}^{(\tau)}), \mathbf{g}^{(\tau)}(r_\tau, r_{\tau+1}) \rangle \cdot h_{r_\tau, r_{\tau+1}}^{(\tau-1)} \\
 &= h^{(\tau)}
 \end{aligned}$$

Ecuación *logsoftmax*:

$$\begin{aligned} p(y_k | \mathbf{x}^{(0)}) &= \log \left(\frac{e^{y'_k}}{\sum_{i=1}^K e^{y'_i}} \right) \quad \forall k \in K \\ &= \log(e^{y'_k}) - \log \left(\sum_{i=1}^K e^{y'_i} \right) \quad \forall k \in K \end{aligned} \quad (\text{Ec. A.3})$$

A.2. LSTM

Como ya se mencionó anteriormente, la idea de una red LSTM surge en sus trabajos anteriores, el paper principal sobre este tipo de redes se publicó un tiempo después [44], y luego fue refinado por sucesivos trabajos [29]. La red llamada **Long Short Term Memory** (LSTM) es parte de un conjunto de redes recurrentes conocidas como *gated RNNs*, éstas también incluyen las red GRU (ver sección A.3). La idea sobre la cual se basan las *gated RNNs* es la de crear caminos a través del tiempo que tengan gradientes que no exploten ni desaparezcan a través del tiempo, permitiendo acumular información, a través de secuencias de larga duración.

La idea de introducir un bucle que se retroalimenta dentro de cada unidad de una red LSTM es lo realmente innovador del modelo original [44], luego se modificó para que los parámetros asociados a ese bucle varíen respecto del contexto en vez de estar fijos [29]. La red LSTM está compuesta de celdas LSTM, equivalentes a las usadas en la red RNN, tienen la misma cantidad de entradas, salidas y pasos pero con más parámetros y con más complejidad en su funcionamiento. La *forget gate* tiene asociada un par de matrices, una para la entrada (\mathbf{U}^f) y otra para el estado oculto del paso anterior (\mathbf{W}^f) y un vector de bias (\mathbf{b}^f):

$$\mathbf{f}^{(t)} = \sigma \left(\mathbf{U}^f \mathbf{x}^{(t)} + \mathbf{W}^f \mathbf{h}^{(t-1)} + \mathbf{b}^f \right) \quad (\text{Ec. A.4})$$

La *external input gate* (Ec. A.5) es idéntica a la *forget gate* pero con su propio conjunto de parámetros, un par de matrices, una para la entrada (\mathbf{U}^g) y otra para el estado oculto del paso anterior (\mathbf{W}^g) y un vector de bias (\mathbf{b}^g):

$$\mathbf{g}^{(t)} = \sigma \left(\mathbf{U}^g \mathbf{x}^{(t)} + \mathbf{W}^g \mathbf{h}^{(t-1)} + \mathbf{b}^g \right) \quad (\text{Ec. A.5})$$

Para calcular el estado interior (Ec. A.6) se usan la *external input gate* (Ec. A.5) y la *forget gate* (Ec. A.4) con el estado interior anterior $\mathbf{s}^{(t-1)}$. Además se utiliza un cálculo idéntico a las otras ecuaciones, usando parámetros iguales, un par de matrices, una para la entrada (\mathbf{U}) y otra para el estado oculto del paso anterior (\mathbf{W}) y un vector de bias (\mathbf{b}), también utilizando una función Sigmoide σ (Ec. 2.3) :

$$\mathbf{s}^{(t)} = \mathbf{f}^{(t)} \cdot \mathbf{s}^{(t-1)} + \mathbf{g}^{(t)} \cdot \sigma \left(\mathbf{U} \mathbf{x}^{(t)} + \mathbf{W} \mathbf{h}^{(t-1)} + \mathbf{b} \right) \quad (\text{Ec. A.6})$$

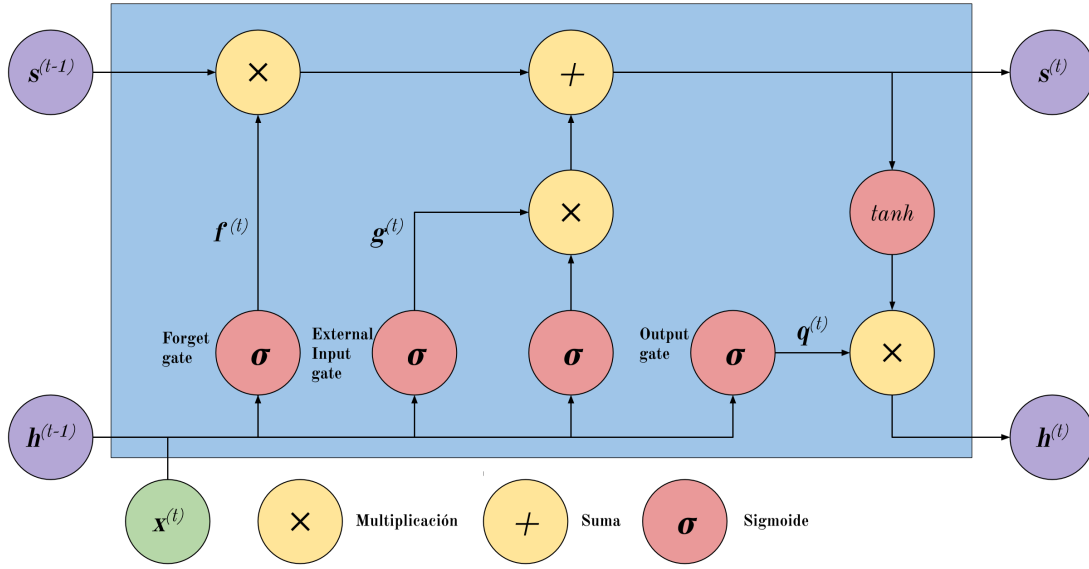


Figura 58: Una celda LSTM [32, 44] equivalente a la celda de una red RNN (Figura 8).

Para calcular la salida $h^{(t)}$ (Ec. A.8), se utiliza la *output gate* (Ec. A.7), usando la misma formulación que las anteriores, con parámetros de la misma forma y la misma función de activación σ (Ec. 2.3). Luego la *output gate* es multiplicada por el estado interior de la celda (Ec. A.6), pero antes se le aplica una función de activación tangente hiperbólica \tanh (Ec. 2.4):

$$q^{(t)} = \sigma \left(U^o x^{(t)} + W^o h^{(t-1)} + b^o + \right) \quad (\text{Ec. A.7})$$

$$h^{(t)} = \tanh(s^{(t)}) \cdot q^{(t)} \quad (\text{Ec. A.8})$$

El principal cambio en la celda LSTM respecto al de una celda RNN es que además de tener la recurrencia común (Figura 6) de una red RNN común, tiene una recurrencia interna (bucle interno) (Figura 58). La entrada de las celda es $x^{(t)}$ y el estado oculto $h^{(t)}$ es la salida de la misma. La parte más importante es la unidad de estado (Ec. A.6), controlado por la *forget gate* (Ec. A.4), está tiene valores restringidos entre 0 y 1 por una función de activación Sigmoide σ (Ec. 2.3). Las redes LSTM han demostrado ser en la práctica mejores que las RNN simples para aprender dependencias a largo plazo.

A.3. GRU

La *Gated Recurrent Unit* (GRU)[13] al igual que la LSTM [44] (ver sección A.2) son parte de las redes recurrentes llamadas *gated RNNs*. Éstas se basan en la idea de atacar los defectos que presentan las RNNs creando caminos a través del tiempo que tengan gradientes que no exploten ni desaparezcan a través del tiempo. La idea detrás de GRU es parecida a la de LSTM, pero con menos parámetros, usando la misma *gate* para controlar simultáneamente el factor de olvidar que tiene la *forget gate* (Ec. A.4) en una LSTM y la decisión de actualizar el estado interno (Ec. A.6).

En el caso de una celda GRU (ver Figura 59), hay una *update gate* (Ec. A.9) y una *reset gate* (Ec. A.10) que tienen la misma forma que las *gates* de una LSTM. Éstas tienen cada una asociada

un par de matrices, una para la entrada ($\mathbf{U}^u, \mathbf{U}^r$) y otra para el estado oculto del paso anterior ($\mathbf{W}^u, \mathbf{W}^r$) y un vector de bias ($\mathbf{b}^u, \mathbf{b}^r$). También usan una función de activación Sigmoide σ (Ec. 2.3) y utilizan como datos, la entrada $\mathbf{x}^{(t)}$ y el estado oculto anterior $\mathbf{h}^{(t-1)}$:

$$\mathbf{g}^{(t)} = \sigma \left(\mathbf{U}^g \mathbf{x}^{(t)} + \sum_j \mathbf{W}^g \mathbf{h}^{(t)} + \mathbf{b}^g \right) \quad (\text{Ec. A.9})$$

$$\mathbf{r}^{(t)} = \sigma \left(\mathbf{U}^r \mathbf{x}^{(t)} + \sum_j \mathbf{W}^r \mathbf{h}^{(t)} + \mathbf{b}^r \right) \quad (\text{Ec. A.10})$$

Para calcular la salida, que es el estado oculto $\mathbf{h}^{(t)}$ de la celda, se utilizan además de la salida de la *update gate* (Ec. A.9) y de la *reset gate* (Ec. A.10), otro conjunto de parámetros, un par de matrices, una para la entrada (\mathbf{U}^h) y otra para el estado oculto del paso anterior (\mathbf{W}^h) que también se hace el producto con el factor de *reset* y un vector de bias (\mathbf{b}^h). La ecuación es la siguiente:

$$\mathbf{h}^{(t)} = \mathbf{g}^{(t-1)} \cdot \mathbf{h}^{(t-1)} + (1 - \mathbf{g}^{(t-1)}) \cdot \sigma \left(\mathbf{b} + \mathbf{U} \cdot \mathbf{x}^{(t)} + \mathbf{W} \cdot \mathbf{r}^{(t-1)} \cdot \mathbf{h}^{(t-1)} \right) \quad (\text{Ec. A.11})$$

La *update gate* (Ec. A.9) y la *reset gate* (Ec. A.10) pueden ignorar partes del estado oculto anterior $\mathbf{h}^{(t-1)}$. La *update gate* puede elegir copiar o ignorar el valor de cualquier dimensión linealmente, mientras que la *reset gate* controla qué partes del estado oculto anterior son tomadas en cuenta para el siguiente estado, introduciendo otro efecto no lineal entre el estado pasado y el futuro.

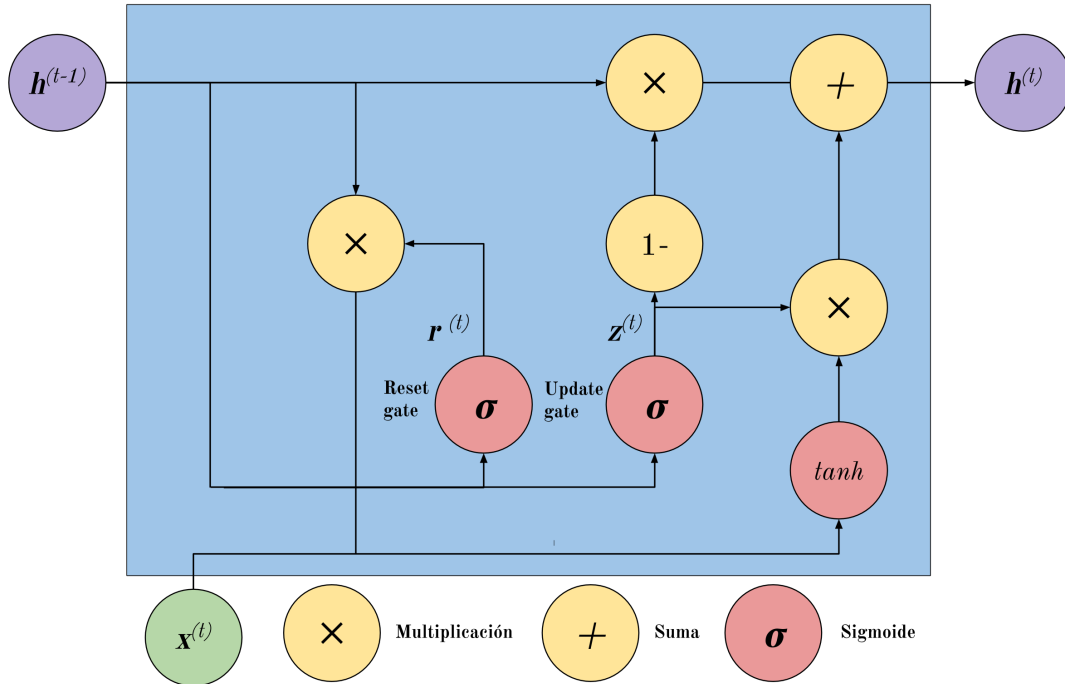


Figura 59: Una celda GRU [13] análoga a una celda RNN (Figura 8). Comparándola con una celda LSTM (ver Figura 58) se puede ver que su cálculo es más simple y requiere de menos parámetros.