UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

EXPERIMENTACIÓN Y COMPARATIVA DE DIFERENTES MODELOS DE REDES NEURONALES ARTIFICIALES PARA EL PROCESAMIENTO DEL LENGUAJE NATURAL

Autor: Gabriel Jiménez Perera

Tutor: Cayetano Guerra Artal

TRABAJO DE FIN DE GRADO – GRADO EN INGENIERÍA INFORMÁTICA

Julio 2017

[Página intencionadamente dejada en blanco]

AGRADECIMIENTOS

Me gustaría empezar esta memoria agradeciendo la colaboración de todas aquellas personas que se han visto implicadas directa o indirectamente en este proyecto.

Empezando por mi tutor, Cayetano Guerra Artal, quien, junto con Francisco Mario Hernández Tejera, me ha acompañado, aconsejado y ayudado en todo lo posible, ofreciendo todo su apoyo en cualquier cosa que le pedía.

No me puedo olvidar de todos aquellos profesores que me han dado su consejo y que siempre estaban dispuestos a escucharme y responder mis inquietudes, como Margarita Díaz Roca, Octavio Mayor González o José Daniel Hernández Sosa.

También tengo que reconocer el mérito de todos aquellos profesores que me han permitido llegar hasta aquí, enseñándome contenidos y valores en diferente medida.

Además de los estrictamente académico, también agradezco enormemente el apoyo de mis amigos, compañeros y compañeros convertidos en amigos, con los que he compartido una etapa muy importante y valiosa de mi vida que, sin ellos, no habría sido tan satisfactoria.

De una manera especial, debo reconocer el mérito de Alberto Casado Garfia, por ayudarme en las prácticas de varias asignaturas y de Laura del Pino Díaz, por su repositorio y consejo, sin los cuáles mi camino no habría sido tan directo.

Finalmente, tengo que agradecer el apoyo de mi familia, que han sido mi pilar fundamental en la vida y me han permitido estudiar, facilitándome todo tipo de medios para lograr esta meta.

[Página intencionadamente dejada en blanco]

RESUMEN

El procesamiento del lenguaje natural ha sido tradicionalmente una tarea compleja y poco trivial en el diseño de algoritmos. Gracias a la inteligencia artificial, se han logrado grandes progresos en este entorno y número de modelos que hacen frente a estos problemas normalmente poco tratables ha ido incrementando.

Este proyecto propone experimentar y comparar tres modelos de redes neuronales artificiales que han tenido bastante éxito en el procesamiento del lenguaje natural: LSTM (Long Short-Term Memory), MemN2N (modelo propuesto por Facebook) y DNC (modelo propuesto por Google). Para esta tarea, estos modelos optimizados han sido adaptados a un ámbito concreto, con el objetivo de comparar los resultados de cada uno.

[Página intencionadamente dejada en blanco]

ABSTRACT

Natural language processing has traditionally been a complex, hardly-trivial task in algorithm design. Thanks to artificial intelligence, great progress has been made in this environment and the number of models that face these usually hardly treatable problems has increasingly grown.

This project proposes experimenting and comparing three artificial neural network models that have had quite accomplishment in natural language processing: LSTM (Long Short-Term Memory), MemN2N (model proposed by Facebook) and DNC (model proposed by Google). For this task, these optimized models have been adapted to a concrete scope, with the objective of comparing the results of each.

[Página intencionadamente dejada en blanco]

ÍNDICE

1 INTRODUCCIÓN

1.1 PRESENTACIÓN DEL PROBLEMA

El procesamiento del lenguaje natural ha resultado ser una tarea poco trivial desde el punto de vista de la generación de algoritmos. Gracias a la inteligencia artificial, se han conseguido grandes avances en este campo. Sin embargo, queda mucho trabajo por hacer para lograr herramientas suficientemente maduras.

El primer problema al que hacer frente desde un punto de vista informático es la representación de la información, pues se debe plasmar el lenguaje natural en vectores de manera que un sistema pueda procesarlos correctamente.

El segundo problema consiste en crear algoritmos capaces de manejar la información, que dada una entrada produzcan la salida deseada.

1.2 ESTADO ACTUAL

Cada año se presentan más y más propuestas de modelos que ofrecen diferentes características y logran mejores resultados en ciertas tareas. Estos modelos pueden estar centrados en la búsqueda de patrones, focalizar la atención en partes del lenguaje, etc.

Las redes LSTM se han utilizado en una gran variedad de problemas con éxito. Su diseño pretende evitar el problema de las dependencias de información a largo plazo, la función para la que está diseñada es recordar datos a largo plazo y hacer predicciones futuras.

El modelo MemN2N de Facebook busca la respuesta a preguntas sobre un entorno concreto sobre el que ha sido entrenado. Funciona centrando la atención en lo que debería ser la respuesta ante la una pregunta concreta. Para ello utiliza una distribución de probabilidad y diferentes espacios de conocimiento.

El modelo DNC de Google utiliza una red LSTM y le añade un módulo de memoria. La red LSTM puede leer y escribir en esta memoria a conveniencia. Esta memoria se diferencia en las memorias clásicas de ordenadores en que no tiene direcciones, el "direccionamiento" que utiliza una distribución de probabilidad.

1.3 OBJETIVOS INICIALES

Se busca replicar diferentes modelos de redes neuronales artificiales actuales y su funcionamiento, concretamente, LSTM, MemN2N y DNC. Para ello, se llevará a cabo parte de la programación y evaluación experimental de tales modelos. Además, se creará un conjunto de datos aplicado a un ámbito concreto y se realizará la comparativa de los diferentes modelos utilizando este conjunto. De forma más específica, las tareas desarrolladas son las siguientes:

Fases	***Duración Estimada*** *(horas)*	***Tareas*** *(nombre y descripción, obligatorio al menos una por fase)*	***Duración Real*** *(horas)*
Estudio previo / Análisis	35	Tarea 1.1: Estudio de redes LSTM, MemN2N y DNC	30
		Tarea 1.2: Estudio del código abierto a utilizar.	25
Diseño / Desarrollo / Implementación	100	Tarea 2.1: Adaptación del código de las redes a utilizar.	45
		Tarea 2.2: Desarrollo del código para la creación del dataset experimental.	15
Evaluación / Validación / Prueba	100	Tarea 3.1: Comprobación, evaluación y comparación de los resultados de rendimiento obtenidos.	120
		Tarea 3.2: Posibles propuestas de mejora en los modelos.	20
Documentación / Presentación	65	Tarea 4.1: Documentación del proceso seguido, así como de las comparativas realizadas.	25
		Tarea 4.2: Preparación de presentación en público y memoria final.	35

1.4 JUSTIFICACIÓN DE LAS COMPETENCIAS ESPECÍFICAS CUBIERTAS

La relación de competencias específicas cubiertas y cómo se han cubierto en este trabajo se presenta en la siguiente tabla:

Competencia	Definición de la competencia	Tarea asociada
CP01	Capacidad para tener un conocimiento profundo de los principios fundamentales y modelos de la computación y saberlos aplicar para interpretar, seleccionar, valorar, modelar, y crear nuevos conceptos, teorías, usos y desarrollos tecnológicos relacionados con la informática.	Análisis de las diferentes estructuras y comparativa entre las mismas.
CP07	Capacidad para conocer y desarrollar técnicas de aprendizaje computacional y diseñar e implementar aplicaciones y sistemas que las utilicen, incluyendo las dedicadas a extracción automática de información y conocimiento a partir de grandes volúmenes de datos.	Construcción y entrenamiento de los modelos.

1.5 APORTACIONES

Con la comparativa entre los tres modelos propuestos, se podrá elegir más fácilmente el adecuado para una tarea dada. Al analizar las ventajas y desventajas de cada modelo ante diferentes entornos y dominios, se realiza una comparativa que serviría para la elección del más adecuado para cada problema y, además, para comprender las diferencias entre las arquitecturas.

Además, con estos experimentos, se podrá entender de una manera más precisa el funcionamiento interno de cada modelo y las ventajas y desventajas de cada uno frente a las dos tareas propuestas.

1.6 METODOLOGÍA DE TRABAJO

1.6.1 MATERIALES PARA EL DESARROLLO

- Equipo de desarrollo
  - Sistema operativo Ubuntu 16.10
  - Python 3.5
  - Tensorflow 1.1/1.2
  - Procesador: Intel Core i7-3770K
  - Memoria RAM: 32 GB
  - Tarjeta gráfica: NVIDIA GTx 690
- Equipo de ejecución 1
  - Sistema operativo Ubuntu 16.04.2 LTS
  - Python 3.5.2+
  - Tensorflow 1.1/1.2
  - Procesadores: 2x Intel Xeon X5675
  - Memoria RAM: 48 GB
- Equipo de ejecución 2
  - Sistema operativo Ubuntu 14.04 LTS
  - Python 3.5.2+
  - Tensorflow 1.1/1.2
  - Procesador: Intel Core i7-5930K
  - Memoria RAM: 16 GB
  - Tarjeta gráfica: 2x NVIDIA GTX 980 Ti
- Código de LSTM (Guerra Artal, 2017)
- Código de MemN2N (Luna, 2017)
- Código de DNC (Samir, 2017)
- bAbI dataset v1.2 (Li, Miller, Chopra, Ranzato, & Weston, 2017)

1.6.2 METODOLOGÍAS DE DESARROLLO

La metodología de desarrollo elegida para este proyecto se basaba en objetivos que se fijaban al cumplir los anteriores. Debido al carácter investigador del proyecto, esta metodología permitía cambios de rumbo constantes, ya que no existía un programa objetivo, sino que la experimentación en sí es el objetivo. Además, al encontrar problemas o elementos interesantes, se podían fijar objetivos nuevos para atajar los problemas e investigar más los casos que se salieran de lo común. De esta forma se consiguió una experimentación extensa centrando el foco de atención en comportamientos inesperado o no intuidos.

Tras la obtención de resultados, se sacaban conclusiones y se discutían. De esta discusión también se fijaban los nuevos objetivos y se comenzaba otra iteración.

2 EXPLICACIÓN DE MATERIAL UTILIZADO

2.1 TENSORFLOW

Tensorflow es una librería de código abierto para la creación de redes neuronales. Creada por Google, reemplaza el lugar de otra librería propia de código cerrado (DistBelief) y existen librerías que añaden capas de abstracción sobre Tensorflow, aunque hay alternativas como PyTorch o Theano. Su código fue liberado a finales de 2015 y actualmente tiene soporte por parte de Google Brain, la comunidad y terceros. La importancia de esta librería es tal, que Google ha desarrollado circuito integrado de aplicación específica personalizado para el aprendizaje automático y adaptado a Tensorflow.

Esta librería ofrece una gran flexibilidad en la creación de redes neuronales y un gran potencial al poder utilizar tarjetas gráficas para ciertas tareas de cálculo intensivo y utilizar librerías de cálculo de alto rendimiento (como BLAS), a costa de cierta complejidad a la hora de ser utilizada. Para atajar esta complejidad, existen frameworks que facilitan la creación de modelos y que, de forma transparente al usuario, utilizan Tensorflow de fondo. Actualmente es una de las librerías más utilizadas y la forma habitual de utilizarla es desde Python, aunque también cuenta con una interfaz para C++.

El funcionamiento básico de esta librería se basa en grafos de flujo de datos. Los nodos del grafo representan operaciones matemáticas, mientras las aristas representan arrays de datos multidimensionales (tensores). Un ejemplo de grafo de Tensorflow puede verse en la **Ilustración 1**.

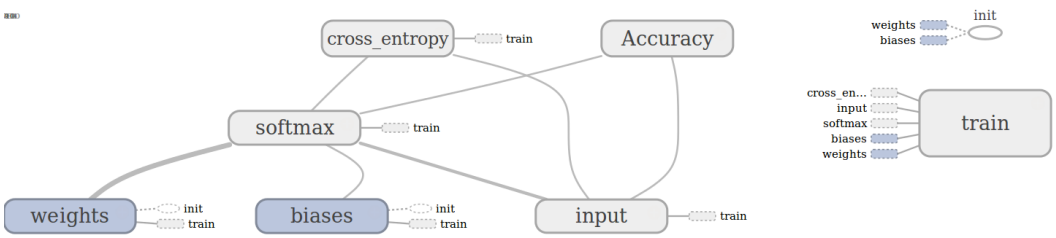

Ilustración 1: Ejemplo de grafo de Tensorflow

Un problema que presenta esta librería es la falta de retrocompatibilidad entre versiones. En este trabajo nos hemos tenido que enfrentar a este problema, lo que hizo tener que adaptar el código de cada modelo a una versión concreta de la librería para poder ejecutarlos correctamente. En este caso la versión estable elegida fue la 1.1.0, aunque también se utilizó la versión 1.2.0 tras su liberación, el 15 de junio, por ser compatible con el código utilizado y ofrecer un rendimiento algo mejor al utilizar las librerías MKL.

Junto con Tensorflow, también se instala una herramienta externa, Tensorboard, que permite la visualización de datos y grafos.

2.2 BABI DATASET

El bAbI dataset de Facebook es un conjunto de datos sintético de tareas pregunta-respuesta creado por Facebook Research (Dialogue Learning With Human-In-The-Loop (forthcoming), arXiv:1611.09823). Este conjunto de datos es de los más usados en la actualidad, ya que cada tarea abarca una habilidad específica esperable de un agente razonador genérico. Cada una de las 20 tareas consiste en 1000 ejemplos de entrenamiento y 1000 ejemplos de test. Las tareas se generan a partir de una simulación de actores y objetos interactuando en un mundo pequeño y cerrado, produciendo texto descriptivo de la escena y parejas de pregunta/respuesta. La supervisión se proporciona en forma de respuestas para cada una de las preguntas junto con la localización de los hechos requeridos para responder la pregunta. La **Ilustración 2** muestra un ejemplo de las 10 primeras tareas.

Task 1: Single Supporting Fact
Mary went to the bathroom.
John moved to the hallway.
Mary travelled to the office.
Where is Mary? A:office

Task 2: Two Supporting Facts
John is in the playground.
John picked up the football.
Bob went to the kitchen.
Where is the football? A:playground

Task 3: Three Supporting Facts
John picked up the apple.
John went to the office.
John went to the kitchen.
John dropped the apple.
Where was the apple before the kitchen? A:office

Task 4: Two Argument Relations
The office is north of the bedroom.
The bedroom is north of the bathroom.
The kitchen is west of the garden.
What is north of the bedroom? A: office
What is the bedroom north of? A: bathroom

Task 5: Three Argument Relations
Mary gave the cake to Fred.
Fred gave the cake to Bill.
Jeff was given the milk by Bill.
Who gave the cake to Fred? A: Mary
Who did Fred give the cake to? A: Bill

Task 6: Yes/No Questions
John moved to the playground.
Daniel went to the bathroom.
John went back to the hallway.
Is John in the playground? A:no
Is Daniel in the bathroom? A:yes

Task 7: Counting
Daniel picked up the football.
Daniel dropped the football.
Daniel got the milk.
Daniel took the apple.
How many objects is Daniel holding? A: two

Task 8: Lists/Sets
Daniel picks up the football.
Daniel drops the newspaper.
Daniel picks up the milk.
John took the apple.
What is Daniel holding? milk, football

Task 9: Simple Negation
Sandra travelled to the office.
Fred is no longer in the office.
Is Fred in the office? A:no
Is Sandra in the office? A:yes

Task 10: Indefinite Knowledge
John is either in the classroom or the playground.
Sandra is in the garden.
Is John in the classroom? A:maybe
Is John in the office? A:no

Ilustración 2: Explicación del bAbI dataset

Por supuesto, en el dataset la información está estructurada de una forma algo menos legible, pues además contiene la información necesaria para la supervisión: los hechos de soporte requeridos para responder la pregunta (que en nuestros experimentos no usamos). En cuanto a contenido, las tareas que utilizamos de este

dataset son muy simples: todas las frases se encuentran en presente simple y siempre se compone de un sujeto, un verbo (que indica movimiento, coger y soltar), y un destino u objeto. Este conjunto de datos utiliza el idioma inglés en la versión utilizada, aunque también existen versiones en hindi y con las letras intercambiadas.

2.3 CODIFICACIÓN DE LAS FRASES

Para poder trabajar con las frases de cualquiera de los conjuntos de datos, estas se deben trasladar a una forma vectorial. Los métodos de codificación de frases en este modelo se basan en la representación "*Bag of words*" (BoW).

2.3.1 BAG OF WORDS

BoW toma la frase $x_i = \{x_{i1}, x_{i2}, \dots, x_{in}\}$, traslada cada palabra a una representación "*one hot*" (un vector de tamaño igual al número de palabras en el vocabulario inicializado a 0 y con un 1 en la posición correspondiente a la palabra) y suma los vectores resultantes: $m_i = \sum_j x_{ij}$. El vector de entrada u, que representa la pregunta, también se traslada como BoW: $u = \sum_j q_j$. El problema que presenta esta traslación es que no se puede capturar el orden de las palabras en la frase, que puede ser importante para algunas tareas.

Para ejemplificar este mecanismo, supongamos un vocabulario = {universidad, llamo, Aristóteles, columna, me, Gabriel, cactus}, que tiene tamaño 7. La representación de las palabras del vocabulario quedaría así:

Palabra	Representación one-hot
universidad	[1,0,0,0,0,0,0]
llamo	[0,1,0,0,0,0,0]
Aristóteles	[0,0,1,0,0,0,0]
columna	[0,0,0,1,0,0,0]
me	[0,0,0,0,1,0,0]
Gabriel	[0,0,0,0,0,1,0]
cactus	[0,0,0,0,0,0,1]

La representación BoW de una frase ("me llamo Gabriel") sería la suma de los vectores de las palabras que se encuentran en dicha frase:

Palabra	Representación
me	[0,0,0,0,1,0,0]
llamo	[0,1,0,0,0,0,0]
Gabriel	[0,0,0,0,0,1,0]
Resultado BoW (+)	[0,1,0,0,1,1,0]

2.3.2 POSITION ENCODING

Para representar el orden de las palabras en las frases (que en nuestros experimentos no se usa), se utiliza una segunda representación que toma la siguiente forma: $m_i = \sum_j l_j \cdot x_{ij}$, donde $\cdot$ es una multiplicación elemento a elemento. l_j es un vector columna con la estructura $l_{kj} = (1 - k/J) - (k/d)(1 - 2j/J)$, asumiendo que se indexa empezando en 1 y siendo J el número de palabras en la frase y d la dimensión de la matriz de traslación. Esta representación, llamada "*Position Encoding*" (PE), hace que el orden de las palabras afecte ahora a m_i. La misma representación se usa para las preguntas y las entradas y salidas de memoria. La forma de este vector puede verse en la **Ilustración 3**.

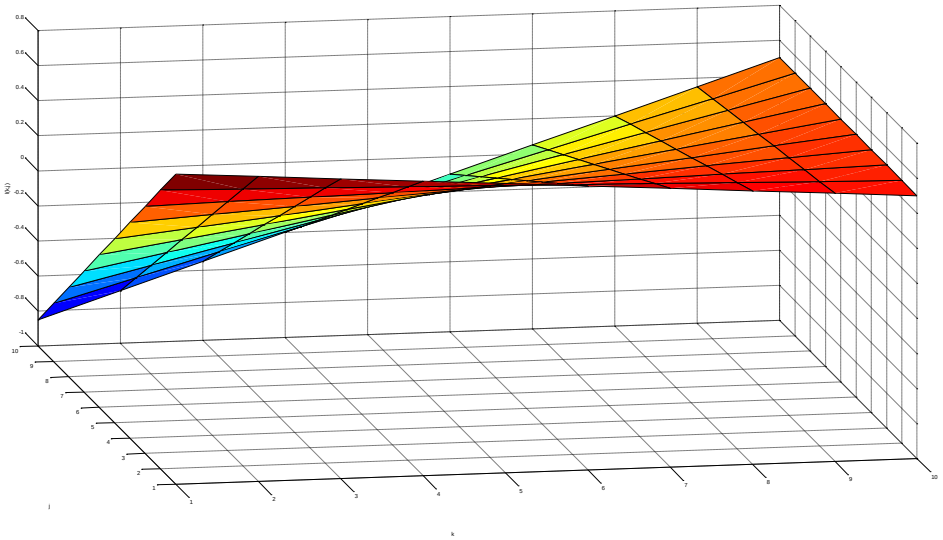

Ilustración 3: Vectores utilizados en Position Encoding

En resumen, Position Encoding asigna diferentes pesos a cada palabra en cada dimensión utilizando una multiplicación elemento a elemento, consiguiendo el efecto de añadir información de situación temporal a cada palabra en cada frase.

3 FRAMEWORKS DE REDES NEURONALES CON MEMORIA

Con este nombre se referencian a las arquitecturas recientes que añaden memoria a la arquitectura clásica de red neuronal y han provocado mayor interés en la comunidad.

3.1 LSTM

Las redes LSTM son un tipo de red recurrente. Las redes recurrentes tienen una analogía con el comportamiento humano: los recuerdos, los pensamientos tienen persistencia. Las redes feed-forward no están preparadas para relacionar entradas en el tiempo de una forma sencilla. Para atajar este problema, las redes recurrentes tienen bucles, por lo que entre las entradas están sus salidas del momento anterior. Este comportamiento puede verse mejor en la **Ilustración 4**.

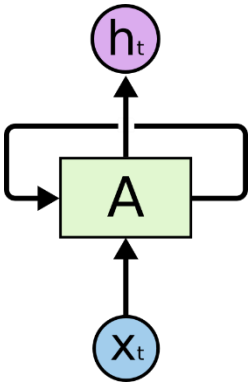

Ilustración 4: Las redes recurrentes contienen bucles

Los bucles aportan una característica muy interesante: poder acceder a información de un momento anterior. Las redes recurrentes pueden verse como varias copias de la misma red, cada una transmitiendo una información a su sucesora. Esto puede verse en la **Ilustración 5**.

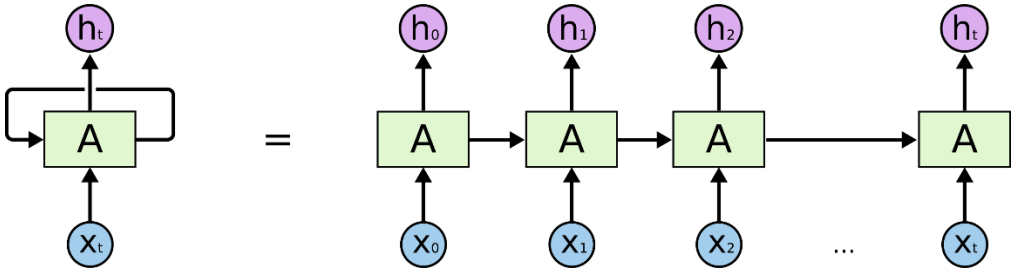

Ilustración 5: Una red recurrente "desenrollada"

La naturaleza de cadena revela que estas redes están ligadas con secuencias y listas. Son las arquitecturas indicadas para el uso de este tipo de datos.

3.1.1 EL PROBLEMA DE LAS DEPENDENCIAS A LARGO PLAZO

Como he indicado, las redes recurrentes están pensadas para casos en los que los datos están relacionados en el tiempo. Sin embargo, esto solo se da cuando el tiempo es suficientemente corto, pues la recurrencia no es una memoria en sí y los datos pasados se pierden muy rápidamente. En la **Ilustración 6: Salida relacionada con entradas pasadas** se ilustra este problema, donde una salida se relaciona con una entrada alejada en el tiempo. La brecha de tiempo entre la salida y la entrada con la que se relaciona es bastante grande y los bucles de la red no son suficiente.

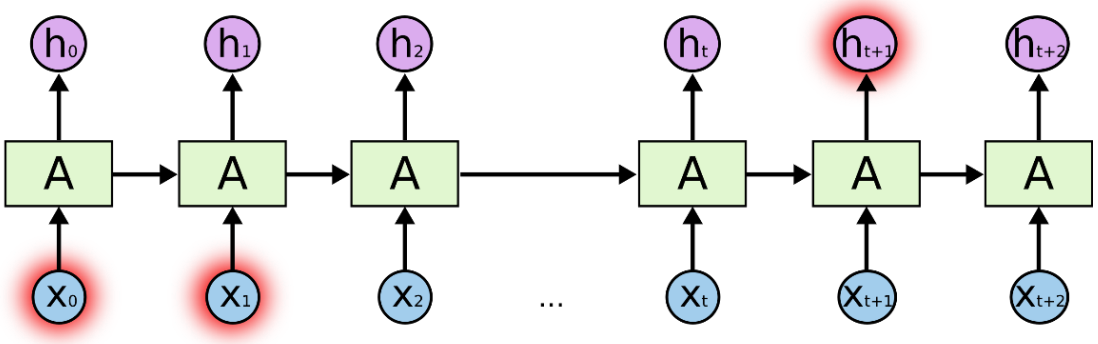

Ilustración 6: Salida relacionada con entradas pasadas

3.1.2 EXPLICACIÓN DEL MODELO

Las redes LSTM (Long Short Term Memory, Hochreiter & Schmidhuber, 1997) son un tipo especial de redes recurrentes capaces de aprender dependencias a largo plazo. Para ello, tienen cuatro redes neuronales de una sola capa interactuando de una forma específica, las líneas negras funcionan a modo de buses de información en la Ilustración 7.

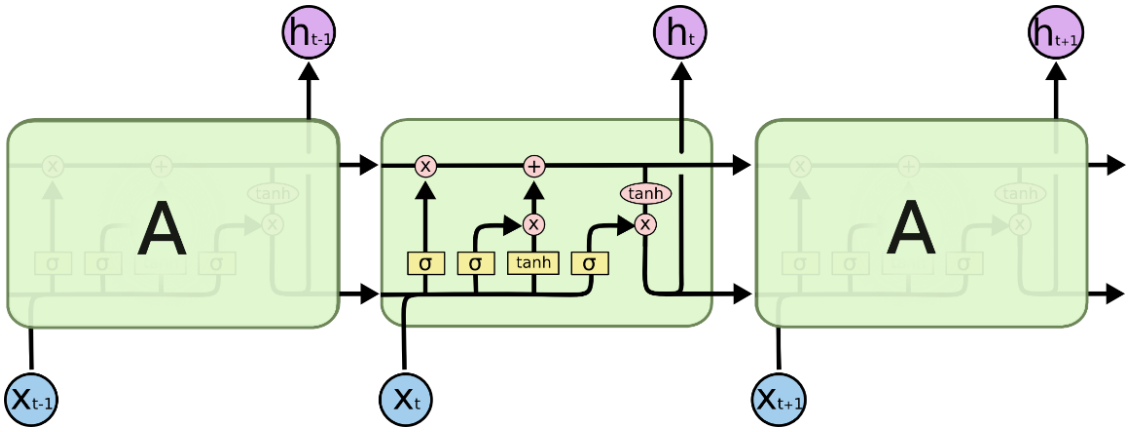

Ilustración 7: Una red LSTM contiene cuatro capas

Las redes LSTM no son tan simples como una red recurrente en las que se basan, pero la idea detrás de ellas no es tan compleja como parece. La parte más novedosa en una LSTM es el estado, destacado en la **Ilustración 8**.

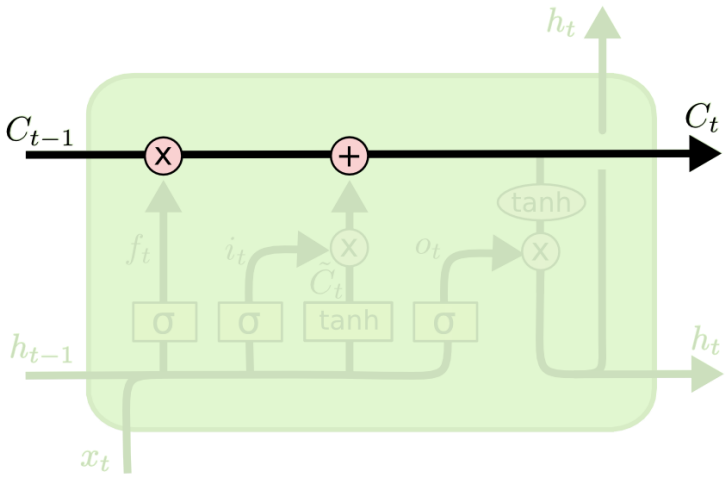

Ilustración 8: Estado de la red LSTM

El estado de una red LSTM es como una cintra transportadora. Va a través de toda la cadena, con solo algunas interaccione. Es muy fácil que la información solo fluya a través del estado sin modificaciones. Las LSTM tienen la habilidad de eliminar o añadir información al estado, pero estas modificaciones están reguladas cuidadosamente por unas estructuras que actúan como puertas.

El primer paso en una LSTM sería decidir qué información del estado se elimina. Esto se hace a través de una red de salida sigmoidal (con salidas entre 0 y 1) que, basándose en la entrada, decide qué información se eliminará, como muestra la **Ilustración 9**.

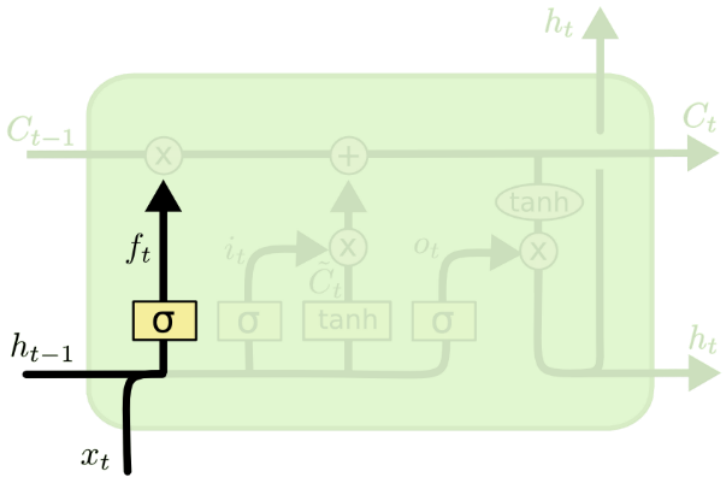

Ilustración 9: Fase de olvido de la información

El siguiente paso sería añadir información al estado. Para ello se parte de la información de entrada y se crea un vector candidato de información nueva, a partir de

una red de salida tangencial (con salidas entre -1 y 1). Por supuesto, no toda la información candidata se memoriza, por lo que se necesita de otra red de salida sigmoidal que decidirá qué información se guardará, como se muestra en la **Ilustración 10**.

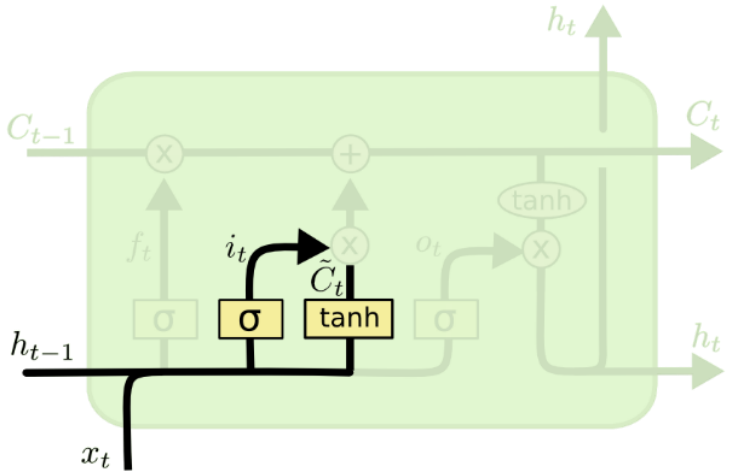

Ilustración 10: Información a añadir al estado

Con los datos obtenidos anteriormente, que decidirán qué información olvidar y cuál recordar, se actualiza el estado. Para ello, se multiplica el estado por el vector obtenido en el primer caso (vector compuesto por valores entre 0 y 1), por lo que se regula qué cantidad de información pasa. Además, se añade la información obtenida en el segundo paso, compuesta por un vector candidato multiplicado por el otro vector obtenido que decidirá en qué grado se guardará, como se enseña en la **Ilustración 11**.

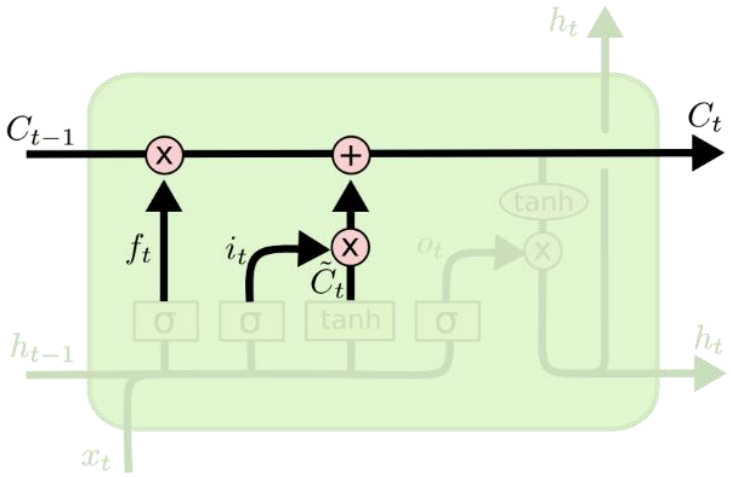

Ilustración 11: Modificación del estado

Finalmente, se genera la salida, que estará basada en el estado, pero de una manera filtrada. La información del estado se pasa por una red de salida tangencial y se multiplica por la salida de una red de salida sigmoidal que depende de la entrada. Este comportamiento se puede ver en la **Ilustración 12**.

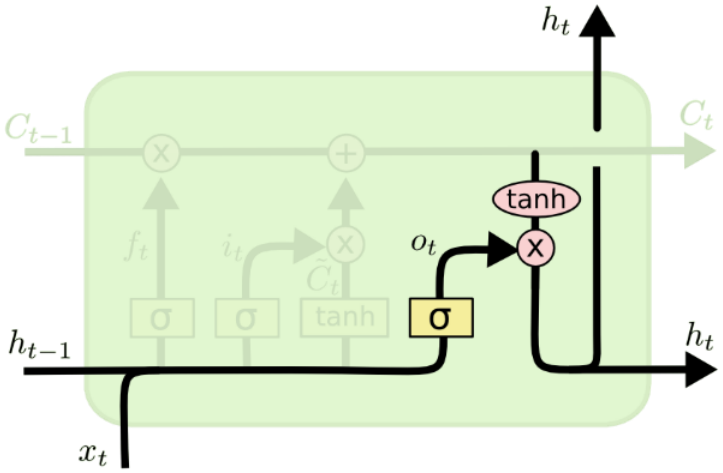

Ilustración 12: Generación de la salida

Con la adición del estado y las modificaciones que se le van haciendo, se pueden recordar datos durante un periodo de tiempo largo, solucionando el problema de las dependencias a largo plazo.

3.1.3 JUSTIFICACIÓN

Las redes LSTM deberían tener un buen comportamiento ya que deberían poder aprender de la historia y, al hacer una pregunta sobre la misma, recoger la información necesaria para responderla. Además, existen trabajos en los que se usa para el tratamiento del habla y para predicción de textos, que son tareas relacionadas con las que se encuentran en el bAbI dataset.

3.2 MEMN2N

La red MemN2N (Sukhbaatar, Szlam, Weston, & Fergus, 2015) está diseñada especialmente para tareas como la que representa el bAbI dataset. Su funcionamiento es relativamente sencillo y se basa en centrar el foco de atención dentro de una frase. A partir de las frases de soporte proporcionadas y una pregunta, esta red busca la frase que debe contener la respuesta y, posteriormente, la extrae para devolverla como la solución.

Para el manejo del foco de atención, este modelo utiliza una función softmax como una distribución de probabilidad. Esta distribución de probabilidad le permite, a partir de una pregunta, seleccionar los datos donde se encuentra la respuesta.

3.2.1 EXPLICACIÓN DEL MODELO

La red MemN2N trabaja con dos memorias, una representa el espacio de preguntas y la otra, el de respuestas. Se basan en centrar el foco de atención dentro de una frase. Este diseño está indicado para problemas como el que presenta el bAbI dataset.

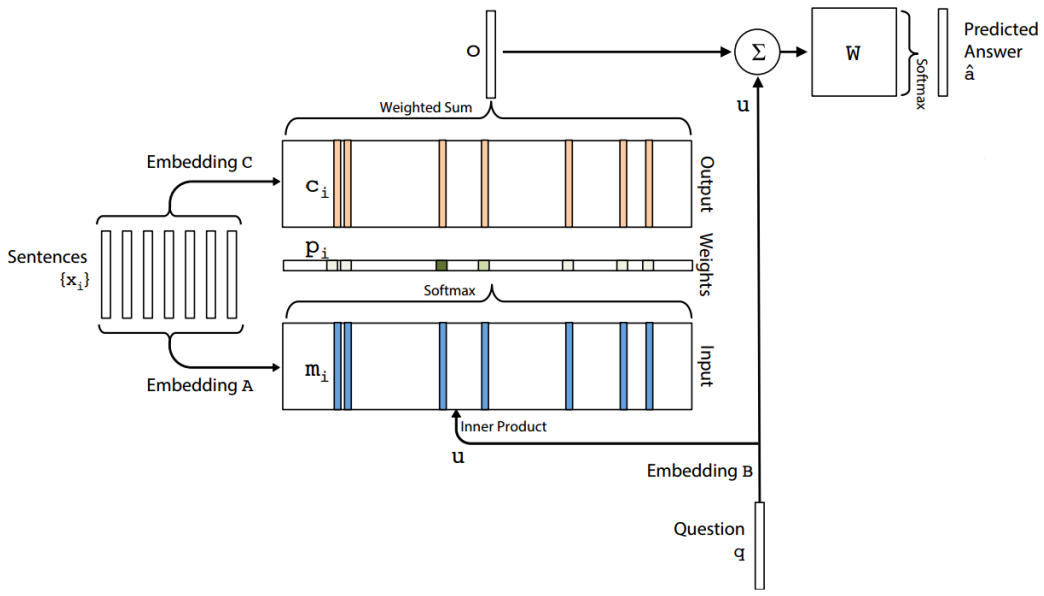

Ilustración 13: Detalle del funcionamiento de una red MemN2N

En la **Ilustración 13** se muestra la estructura de esta arquitectura. *Embedding* son matrices por las que se multiplican las entradas para pasar los datos a los diferentes espacios. Por tanto, la codificación de las frases en este modelo se realiza de forma similar a la explicada en el punto **Codificación de las frases**, con la diferencia que se utilizan matrices cuyos valores se aprenden para realizar una traslación a los diferentes espacios representados como memorias. De esta forma, resultan $m_i = \sum_j Ax_{ij}$, $c_i = \sum_j Cx_{ij}$ y $u = \sum_j Bq_j$.

Además, para representar el orden de las frases dentro de una historia, se modifica el vector de memoria de la siguiente forma: $m_i = \sum_j Ax_{ij} + T_A(i)$, donde $T_A(i)$ es la i-ésima fila de una matriz especial T_A que codifica la información temporal. La traslación de salida se realiza de la misma forma con una matriz T_C, de forma que $c_i = \sum_j Cx_{ij} + T_C(i)$. Tanto T_A como T_C son aprendidas durante el entrenamiento. También están sujetas a las mismas restricciones que A y C. Los hechos se indexan en orden inverso, reflejando su distancia relativa de la respuesta, de forma que x_i es el último hecho de la historia.

En este diseño, *Embedding A* y *Embedding C* convertiría las frases de soporte a espacios de soporte y respuesta, respectivamente. La pregunta se convertiría al espacio de soporte mediante *Embedding B*, lo que permitiría elegir los datos que están más relacionados con la pregunta. Con esta información se crearía una distribución de probabilidades (pesos) mediante la capa *Softmax*. El espacio de respuestas se somete, entonces, a una suma pesada con los pesos obtenidos anteriormente. Con esta suma pesada, que contiene la información de soporte necesaria en el espacio de respuestas y la pregunta en sí, en el espacio de soporte, se realiza una suma y se multiplica por una matriz de pesos, cuyo resultado se pasa por otra capa *Softmax* para generar la respuesta.

Por tanto, la parte fundamental del proceso es la generación de la distribución de probabilidades en la primera capa *Softmax*. Esta se encarga de centrar el foco de atención en la respuesta basándose en la información de soporte y la pregunta. El resto del proceso simplemente le prepara la información para que el proceso sea efectivo, pue simplemente convierten la información a un espacio determinado.

Además, para el tratamiento de problemas complejos, este modelo utiliza un número de iteraciones o *hops*, de forma que la salida de la primera iteración es la entrada de la segunda, y así se itera hasta conseguir la última salida que sería la deseada. Este comportamiento puede verse en la **Ilustración 14**.

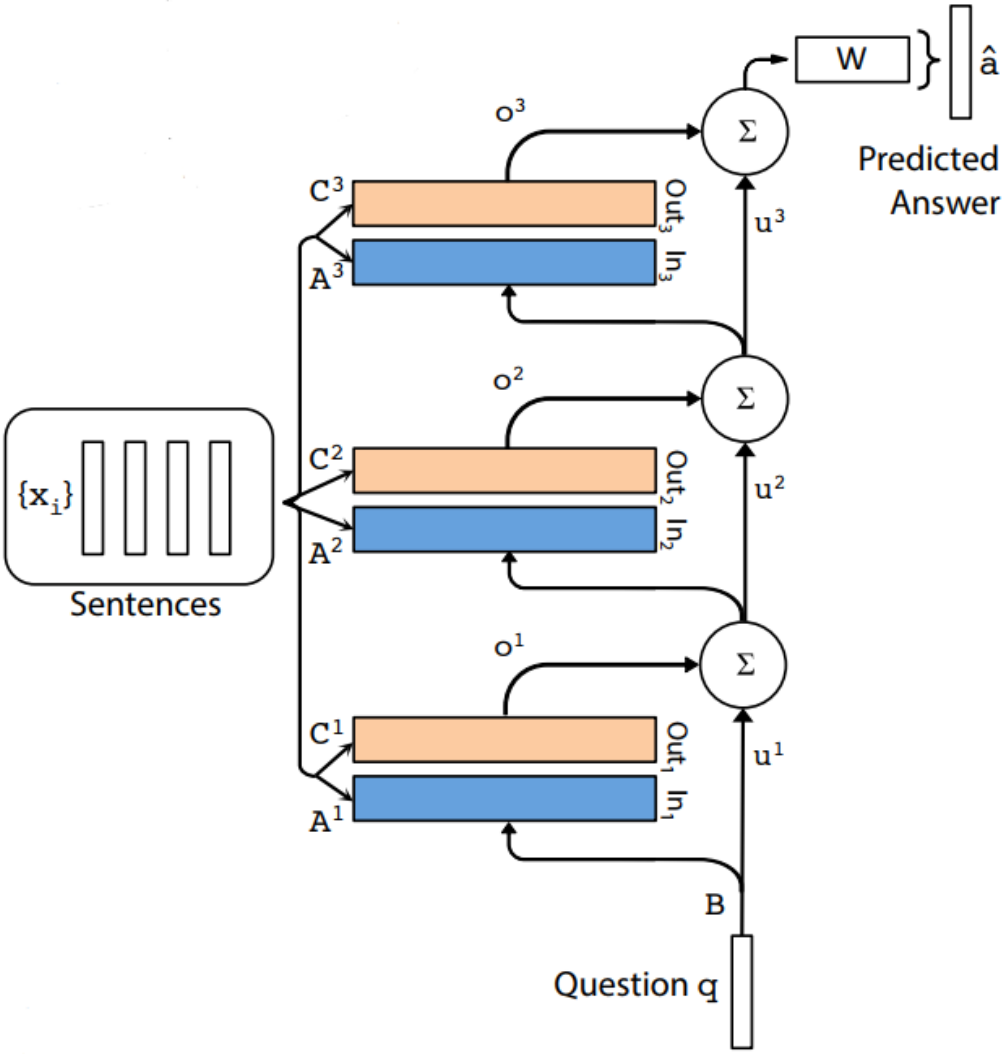

Ilustración 14: Hops en la red MemN2N

De esta manera se consigue dividir un problema demasiado complejo para un solo hop, que sería el modelo simple, en problemas más pequeños. De forma general, al aumentar el número de hops utilizado, se mejoran los resultados obtenidos.

3.2.2 JUSTIFICACIÓN

Este modelo se escoge porque está creado de forma específica para tareas como la del bAbI dataset, ya que resuelve correctamente el problema de la dependencia a largo plazo al tener memoria y se centra en buscar la respuesta a la pregunta basándose en las frases de soporte.

3.3 DNC

Las redes neuronales artificiales se utilizan para procesamiento sensorial, aprendizaje de secuencias y aprendizaje por refuerzo, pero están limitadas en su habilidad de representar variables y estructuras de datos y para almacenar datos durante mucho tiempo, debido a la falta de una memoria externa.

Los ordenadores neuronales diferenciables (Graves, y otros, 2016), también llamados DNC por sus siglas en inglés, se basan en las redes LSTM explicadas anteriormente, pero tiene una característica añadida que le aporta una gran potencia: memoria accesible. De esta forma se pretende que la red LSTM actúe de controlador y, mediante unas cabezas de lectura y escritura, pueda acceder en cada momento a la información específica que requiera en cada momento. Una forma temprana de DNC, la máquina neuronal de Turing, tenía una estructura similar; pero sus métodos de acceso a memoria eran más limitados.

La motivación inicial era el tratamiento de grafos, es decir, poder mostrar un grafo a la red y hacer preguntas sobre el mismo. Uno de los mayores problemas sería la presentación del grafo, pues son de tamaño variable y la presentación secuencial en una red LSTM normal acabaría por saturar su sistema de memoria. La motivación de la memoria externa es solventar esta problemática, de forma que la red puede escribir los datos según los recibe y, a la hora de buscar la respuesta a una pregunta, puede leer libremente por toda la información proporcionada anteriormente.

3.3.1 EXPLICACIÓN DEL MODELO

Por tanto, los DNC consisten en una red neuronal que puede leer y escribir de una memoria matricial externa, de manera análoga a la memoria de acceso aleatorio de un ordenador convencional. Con esta característica, puede usar la memoria para manipular estructuras de datos complejas como un ordenador, pero aprender a hacerlo a partir de la información en sí como una red neuronal. Cuando se utiliza aprendizaje supervisado, se puede enseñar a un DNC a responder preguntas sintéticas diseñadas para emular razonamiento y problemas de inferencia en lenguaje natural. Esta estructura puede realizar otras tareas como path-finding, resolución de puzles de bloques… En resumen, las DNC pueden resolver tareas estructuradas complejas que son inaccesibles a redes neuronales sin una memoria de lectura-escritura externa. Sin embargo, la motivación inicial era buscar una estructura capaz de responder preguntas sobre un grafo, para lo que se le daría información secuencialmente y, al hacer la pregunta, se necesitaría revisar la información dada.

A partir de la **Ilustración 15**, podemos observar que el sistema se compone, principalmente, de una red (controlador), una memoria, cabezas de lectura y escritura y una representación de uso de memoria y enlaces temporales. El controlador sería una red recurrente (en este caso, pues se pueden utilizar otro tipo de redes) y se encargará de leer la entrada y producir la salida, para lo que puede leer y escribir cuantas veces estime necesario en la memoria y acceder a los datos de uso y enlaces temporales.

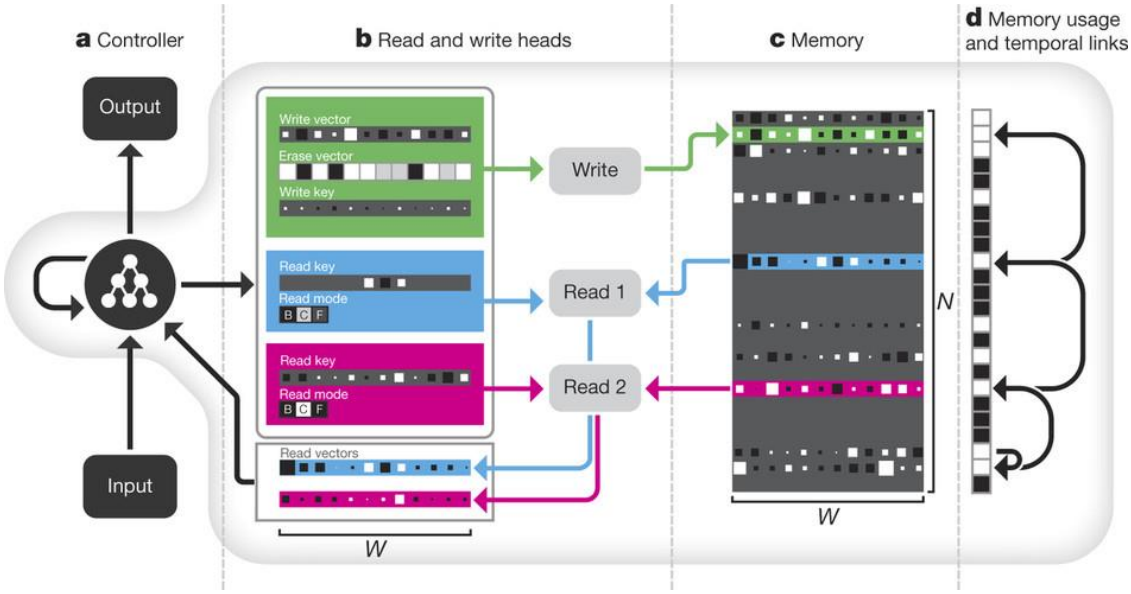

Ilustración 15: La arquitectura DNC

En contraposición a los ordenadores, que utilizan direcciones de memorias únicas para acceder a contenidos de memoria, DNC utiliza mecanismos de atención diferenciables para definir distribuciones sobre las filas de memoria. Estas distribuciones representan el grado en que cada localización está involucrada en cada operación de lectura/escritura. El vector de lectura obtenido es una suma pesada sobre las localizaciones de memoria. De forma similar, las operaciones de lectura utilizan primero un vector de borrado y después se añade un vector de escritura.

Para utilizar la memoria externa, se dispone de una cabeza escritora (la verde en la ilustración, pero podría haber más de una) y varias cabezas lectoras (dos en este caso, la azul y la rosa).

La cabeza escritora define un vector de escritura y otro de borrado que se usan para editar la matriz de memoria; el vector de escritura define la información que se desea escribir y, el de borrado, la información que se añade al vector de escritura para sobrescribir los datos que se encontraban anteriormente en la memoria. Además, se utiliza una clave de escritura se usa en la búsqueda de contenido para encontrar direcciones de escritura previa a editar. La clave de escritura puede contribuir a definir unos pesos que centran de forma selectiva la operación de escritura en las diferentes filas o localizaciones de la matriz de memoria.

Las cabezas de lectura pueden usar puertas llamadas modos de lectura para cambiar entre búsqueda de contenido y lectura de direcciones secuenciales hacia adelante o atrás según el orden en que fueron escritas (utilizando la información de uso de memoria). La información leída serán datos de entrada del controlador en la siguiente iteración.

El vector de uso guarda qué localizaciones han sido utilizadas, y una matriz de enlaces temporales guarda el orden en que las localizaciones fueron escritas. Esto es útil, por ejemplo, para casos en los que una secuencia de instrucciones debe almacenarse y devolverse en orden.

Sin embargo, el tamaño de la matriz de enlaces es NxN, siendo N el número de filas de la memoria de DNC. Esto quiere decir que requiere $O(N^2)$ recursos tanto de memoria como de cómputo. Esto quiere decir que el coste aumenta rápidamente según aumenta el número de posiciones de memoria. Afortunadamente, la matriz de enlaces es normalmente muy escasa y puede aproximarse a un coste computacional $O(N \log N)$ y $O(N)$ de memoria sin una pérdida perceptible de rendimiento.

3.3.2 JUSTIFICACIÓN

Este modelo se ha elegido porque posee las características de las LSTM, solo que con un potencial mucho mayor. Además, su memoria externa le permite atajar el problema de la dependencia a largo plazo de una manera más potente y también le permite el manejo de estructuras internas que reflejen características del lenguaje humano.

4 DATASET PROPIO

Para realizar pruebas en un entorno concreto, se ha creado un dataset propio. El ámbito elegido para el dataset es la familia, por lo que los hechos consisten en relaciones entre familiares y las preguntas piden inferir una relación. Esta tarea es más compleja que la encontrada en el bAbI dataset, ya que se pide una inferencia (a la hora de obtener la respuesta, se requiere buscar el complemento de una relación dada, por ejemplo, se busca inferir que la relación complementaria de "padre" es "hijo"). En este dataset se utiliza un modelo simple de frase que indica una relación, de forma que se proporcionan aristas del grafo de relaciones, formadas por dos personas y la relación que las une.

Pese a estar en español, este dataset está simplificado, pues solo se utilizan nombres masculinos y las frases siempre tienen la misma estructura. Utilizar solo un género en los nombres nos permite reducir el vocabulario, pues nos permite prescindir de palabras de relaciones equivalentes ("padre" y "madre"). Además, al utilizar la misma estructura de hecho se reduce la complejidad de las frases para una red; como experimentación futura se propone añadir otras estructuras como "resulta que A es X de B" o "A es X de B según C", que son frases más cercanas al lenguaje usado por los humanos.

Como modelo sobre el que basar los hechos, se utiliza una familia tipo como la que se puede observar en la **Ilustración 16**. De esta forma, se sabe a priori las relaciones entre los distintos nodos y solo hay que nombrar cada uno para poder obtener hechos.

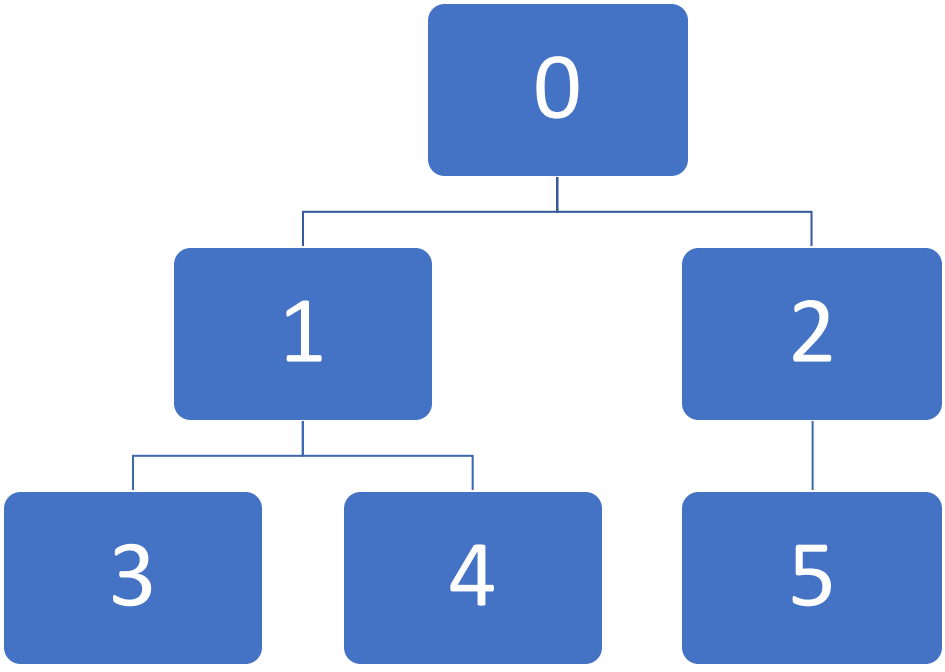

Ilustración 16: Modelo de familia para el dataset propio

Como parámetros, se puede elegir el número total de historias, un rango para el número de hechos por historia, y para qué dificultades se desea hacer un conjunto de datos. La dificultad será el número de hechos que se deberán consultar para poder inferir la respuesta. La forma del dataset se muestra en la **Ilustración 17**. Se garantiza que el número de hechos mínimo para poder inferir la respuesta es el dado por el usuario, ya que los familiares implicados en la relación solo se usan en los hechos de soporte.

```
Franco es hermano de Javier
Javier es tío de Cristian
Javier es tío de Cristian
Cristian es primo de Gabriel
Franco es hermano de Javier
Gabriel es primo de Cristian
Javier es hermano de Franco
Lucas es abuelo de Sebastian
Franco es hermano de Javier
Javier es hermano de Franco
¿Qué es Sebastian de Lucas? nieto
```

Ilustración 17: Ejemplo del dataset propio

5 METODOLOGÍA DE PRUEBAS UTILIZADO

Debido a las diferentes versiones sin retrocompatibilidad de Python y de Tensorflow, se ha adaptado el código de los tres modelos a la versión 3.5 de Python (compatible también con la versión 3.6) y 1.1 de Tensorflow. Entre las versiones 1.1 y 1.2 de Tensorflow existe una retrocompatibilidad completa en los módulos utilizados en este trabajo. Además, Tensorflow 1.2 puede compilarse con las librerías MKL, por lo que, desde que salió esta última versión (15 de junio), se procedió a su uso por ofrecer un rendimiento mayor.

Una vez adaptado el código, se procedió a hacer pruebas con el mismo. Estas consistían en ejecuciones de cada modelo, adaptando los parámetros de cada uno a nuestros problemas y buscando diferencias en el comportamiento según los parámetros utilizados para decidir qué parámetros serían idóneos.

Para automatizar un poco la toma de pruebas, se añadió código adicional que guardaba los resultados en formato CSV o redirigía la salida a un fichero de texto que se procesaba posteriormente. Además, se utilizaban bucles o scripts para poder tomar múltiples muestras a partir de una sola ejecución.

Por otro lado, para conseguir una mayor veracidad de las pruebas, se utilizaban ejecuciones simultáneas en varios ordenadores, para evitar posibles resultados que se salieran de un margen de error.

Para la obtención de gráficas, se utilizaron programas de ofimática y Tensorboard, una herramienta incluida en el entorno de Tensorflow.

6 RESULTADOS OBTENIDOS

6.1 LSTM

El uso de esta red supone un problema: el sobreajuste; es decir, que la red se acostumbra a los datos de entrenamiento y se los memoriza, pero no aprende a generalizar, que es lo que se espera. Esto provoca que, al presentar datos nuevos, los resultados no son tan buenos como al presentar datos ya vistos; lo que obliga a utilizar técnicas de regularización. Además, esta red no está tan centrada en los problemas para los que se ha utilizado (como sí lo está MemN2N), pues es más general, y no ofrece la potencia de una memoria externa (en contraposición a DNC). Por ello, se han realizado pruebas con LSTM utilizando dropout como técnica de regularización y varias capas.

En este caso, se han realizado pruebas sobre el bAbI dataset con la tarea de un hecho de soporte. La limitación a esta tarea solamente se debe a que hay muchas variantes de las redes LSTM y consideramos más relevante hacer pruebas sobre dichas variantes que sobre varias tareas diferente. Más aun teniendo en cuenta que, pese a ser entrenada con la tarea más sencilla, se obtienen los peores resultados.

En primer lugar, vamos a ver cómo se comporta una red LSTM utilizando una capa y dropout al 50%, por lo que en cada fase de entrenamiento solo la mitad de las neuronas aprenderán. Durante la fase de aprendizaje, se llega a una precisión que ronda el 75% como se muestra en la **Ilustración 18**. A priori, estos datos son bastante relevantes, pero entonces los contrastamos con los datos de validación y obtenemos los datos de la **Ilustración 19**. Como se puede ver, pese a utilizar una regularización bastante severa, se produce un sobreentrenamiento demasiado elevado, pues apenas se pasa un 31% de precisión.

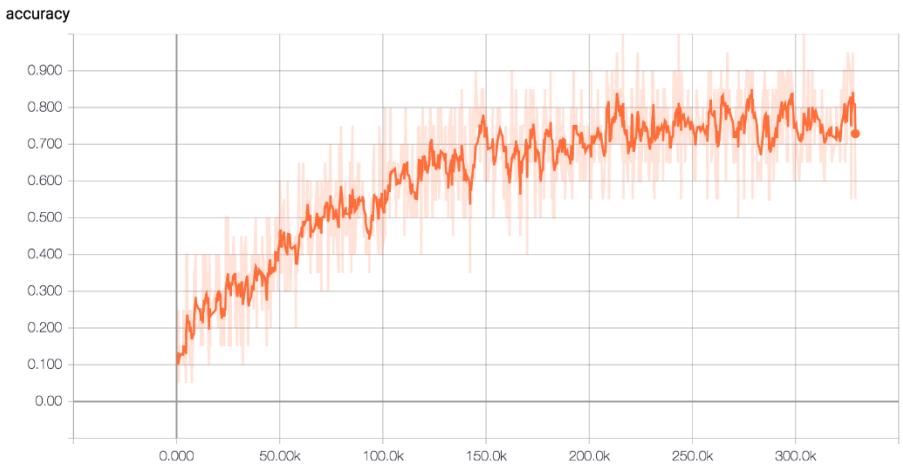

Ilustración 18: Evolución LSTM una capa y dropout 0.5

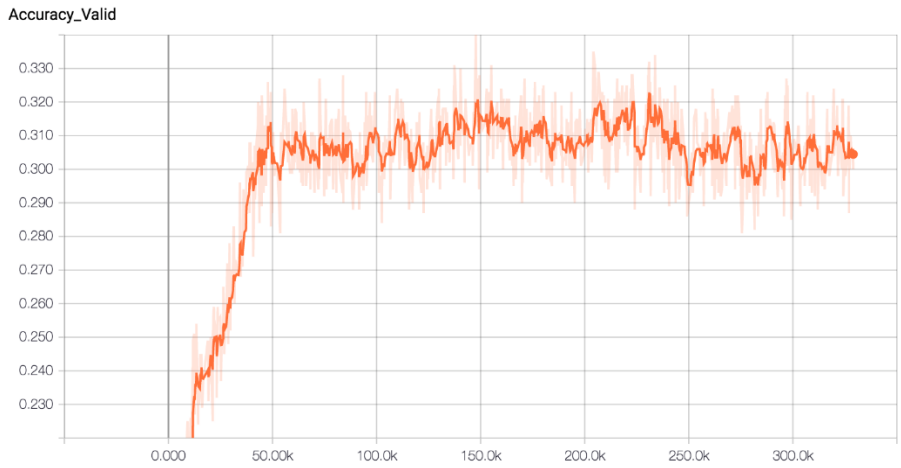

Ilustración 19: Validación LSTM una capa y dropout 0.5

En segundo lugar, se intentó buscar una solución mejor añadiendo una segunda capa. Al añadir otra capa, se espera que una red pueda resolver tareas más complejas, pero necesita más tiempo de entrenamiento. En nuestro caso, se espera que añadir otra capa resuelva mejor nuestro problema si este es muy complicado para dos capas. De nuevo se produce mucho sobreentrenamiento, pero el resultado final es mejor, tal y como se puede ver comparando la **Ilustración 20** con la **Ilustración 21**. Nótese la reducción en el número de iteraciones utilizado entre la prueba anterior (325k) y esta (200k).

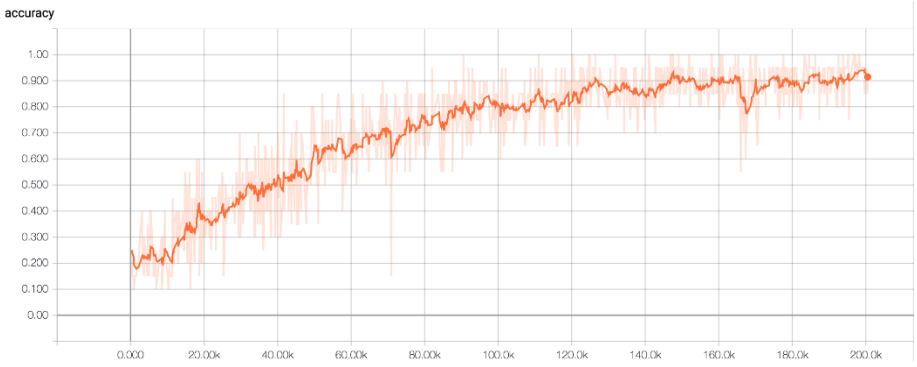

Ilustración 20: Evolución LSTM 2 capas y dropout 0.5

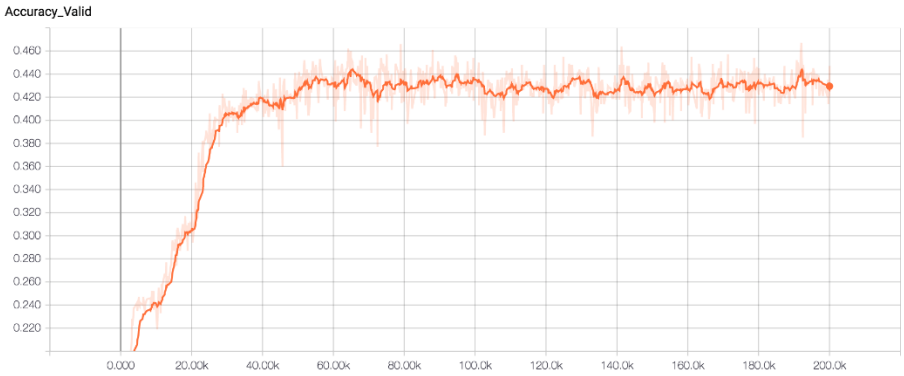

Ilustración 21: Validación LSTM 2 capas y dropout 0.5

En tercer lugar, se probó con 3 capas. Siguiendo con la dinámica anterior, vemos un entrenamiento con buenos resultados a priori **Ilustración 22**, aunque algo más bajos ya que no se llega a un número de iteraciones tan grande, pues ya se empieza a ver cómo la precisión se empieza a aumentar más lentamente y solo se pretende ver el resultado de añadir una tercera capa a la red. Sin embargo, vemos en la **Ilustración 23** que el resultado de validación también es peor al añadir la tercera capa, lo cual es un resultado curioso y no esperado.

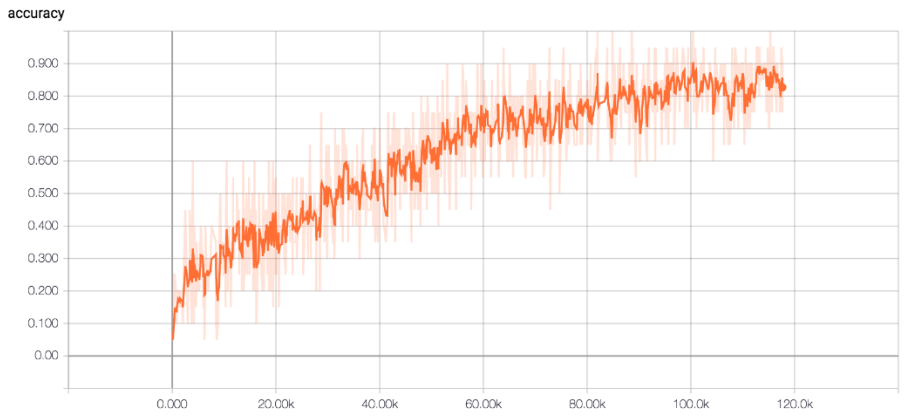

Ilustración 22: Evolución LSTM 3 capas y dropout 0.5

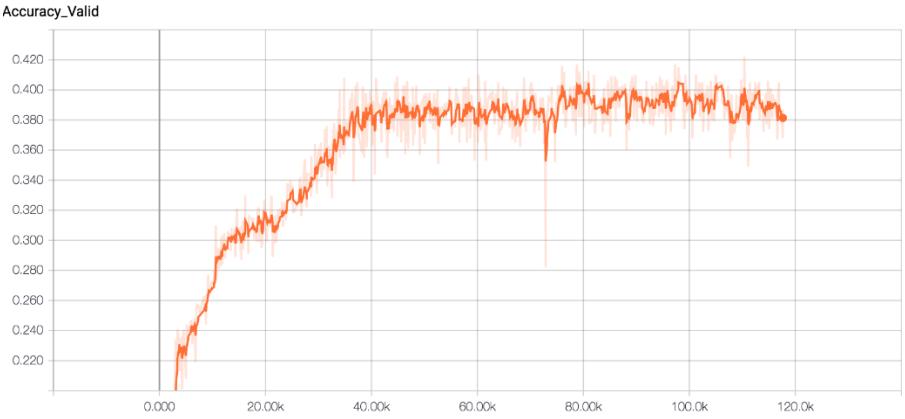

Ilustración 23: Validación LSTM 3 capas y dropout 0.5

En cuarto lugar, y debido al resultado anterior, se hizo una última prueba añadiendo una cuarta capa. De esta forma se pretende ver si añadir capas siempre empeora el resultado o nos habíamos encontrado ante un resultado anómalo. En este caso vemos un entrenamiento bastante progresivo en la **Ilustración 24**. Sin embargo, la precisión en el conjunto de validación vuelve a mejorar, como se muestra en la **Ilustración 25**. Por lo que el resultado anterior se considera anómalo.

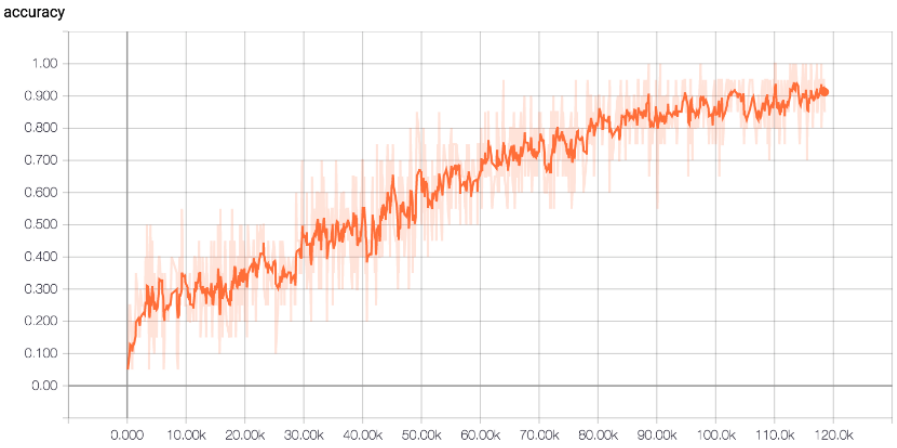

Ilustración 24: Evolución LSTM 4 capas y dropout 0.5

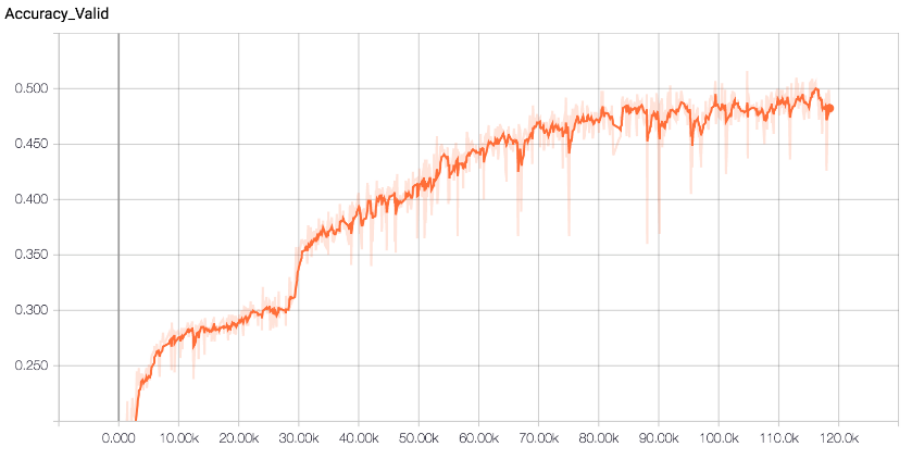

Ilustración 25: Validación LSTM 4 capas y dropout 0.5

Con este último resultado, vemos que el hecho de añadir capas tiende a mejorar el resultado, aunque no en todos los casos. Sin embargo, nos seguimos encontrando ante el problema del sobreentrenamiento y de la precisión, que apenas llega a rozar el 50% en el mejor de los casos. Por ello, como última prueba, se utilizó una LSTM bidireccional (que es más potente en algunos problemas). En este caso se mantuvieron las 4 capas y el dropout al 50% y se redujo levemente el número de iteraciones. Como resultado vemos que el entrenamiento vuelve a mostrar la dinámica seguida anteriormente en la **Ilustración 26**, pero el resultado en la validación (**Ilustración 27**) es peor que utilizando una LSTM normal; por lo que las LSTM bidireccionales no aportan mejoras sino que, al contrario, empeoran los resultados.

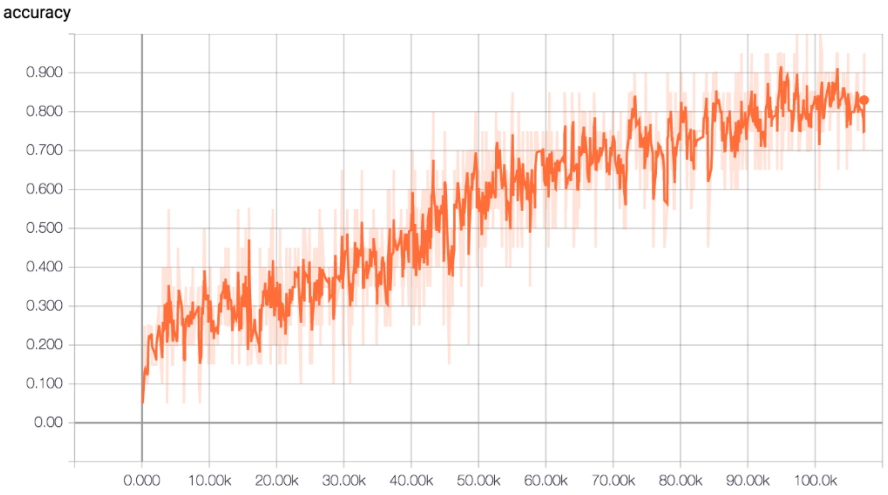

Ilustración 26 Evolución LSTM bidireccional 4 capas y dropout 0.5

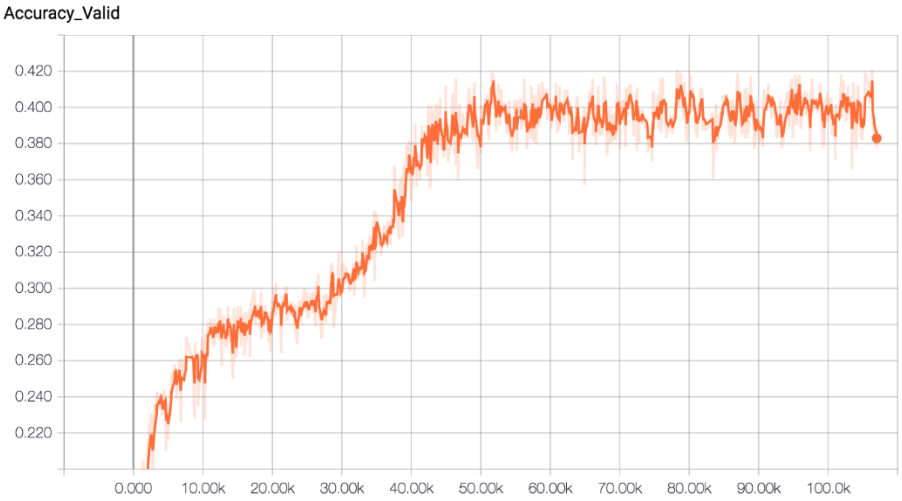

Ilustración 27 Validación LSTM bidireccional 4 capas y dropout 0.5

6.2 MEMN2N

Esta red ofrece los mejores resultados, lo cual era esperable ya que está diseñada especialmente para el tipo de tareas utilizado. En este modelo nos hemos centrado en modificar el número de *hops* utilizados en cada tarea y esto nos ha permitido mejorar los resultados más bajos.

Como se puede ver en la **Ilustración 28**, las tareas de bAbI se resuelven con bastante facilidad en este modelo; alcanzando una precisión casi completa para las tareas 1 y 2 y mayor del 80% para la tarea 3. Sin embargo, en el dataset propio la tarea 1 es la que se resuelve con buenos resultados mientras que las tareas 2 y 3 no consiguen llegar a unos resultados aceptables, pues no pasan del 60% de precisión.

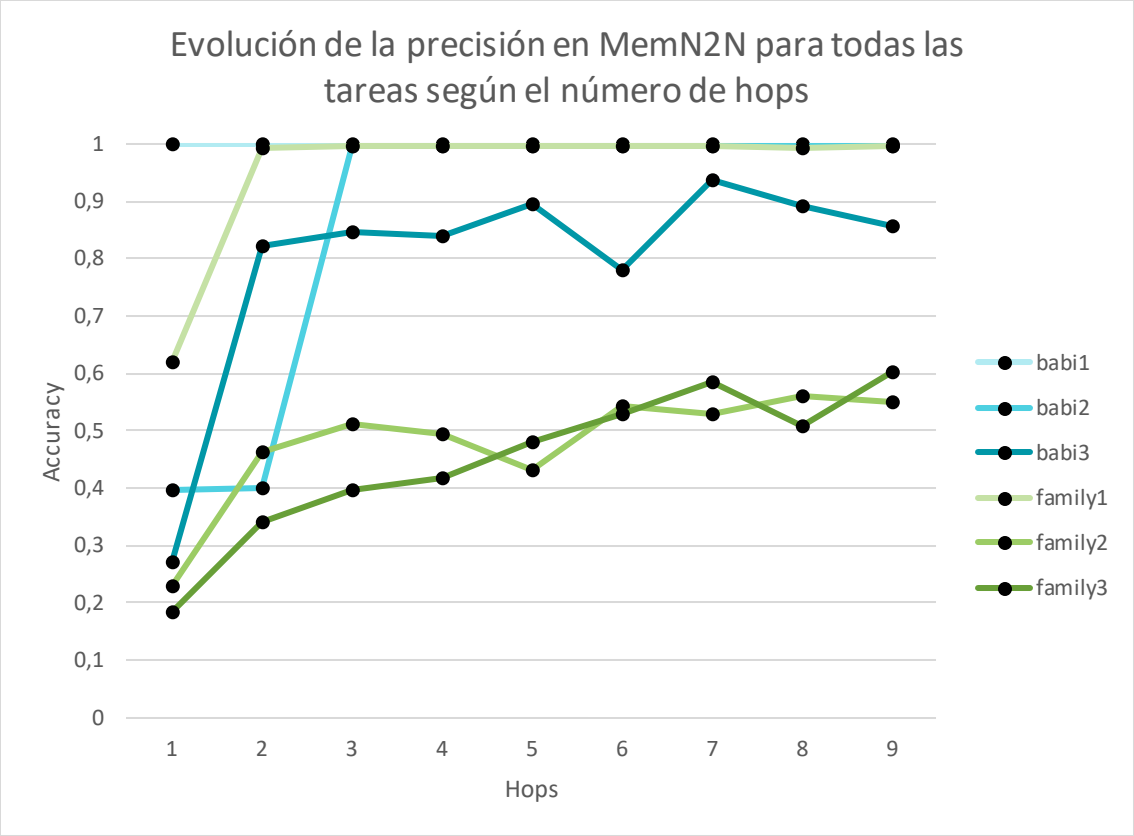

Ilustración 28: Evolución de la precisión según el número de hops utilizados

Esta diferencia de resultados puede deberse a que, en bAbI, la respuesta es directa y de izquierda a derecha, p.e.: "María está en el salón, ¿dónde está María?". Mientras que, en el dataset propio, la respuesta esperada requiere de una inferencia de derecha a izquierda, p.e.: "Pedro es padre de Pablo, ¿qué es Pablo de Pedro?". En el dataset propio se requiere buscar la frase de soporte que contenga a Pablo como objeto, a Pedro como sujeto y se deberá inferir la relación complementaria de "padre"; mientras que, en bAbI, solamente se requiere buscar el sujeto y responder con el objeto. Debido a que en las tareas 2 y 3 se requiere buscar en varios hechos de soporte, la complejidad extra se multiplica, lo que nos lleva a los resultados mostrados.

En cuanto a requisitos de tiempo, las tareas del dataset bAbI requerían más tiempo para llevar a cabo todas las iteraciones. En el caso de dos hechos de soporte y 9 hops, el tiempo requerido para el entrenamiento fue de unos 17 minutos, mientras que, para el dataset propio, apenas se requirieron unos 7. Sin embargo, también se puede ver que para las tareas sencillas con, como mucho, 3 hops, ya se consiguen los mejores resultado o resultados muy aproximados a los mejores. Sin embargo, a medida que aumenta la complejidad, añadir hops mejora los resultados en cantidades apreciables pero inferiores a las deseables.

6.3 DNC

Este modelo tiene una complejidad bastante mayor que los dos anteriores. Se espera que sea más flexible y aplicable a más problemas, pero carece de la especialización en un problema concreto. Debido a la complejidad de este modelo, el entrenamiento es mucho más costoso en tiempo y no ofrece unos resultados tan buenos como los dos anteriores. Como se puede ver en la **Ilustración 29**, para las tareas de bAbI se consiguen mejores resultados y, curiosamente, en el dataset propio las preguntas que requieren de 3 hechos de soporte ofrecen un error menor que las que requieren de un solo hecho de soporte, no hemos podido encontrar una explicación para este comportamiento.

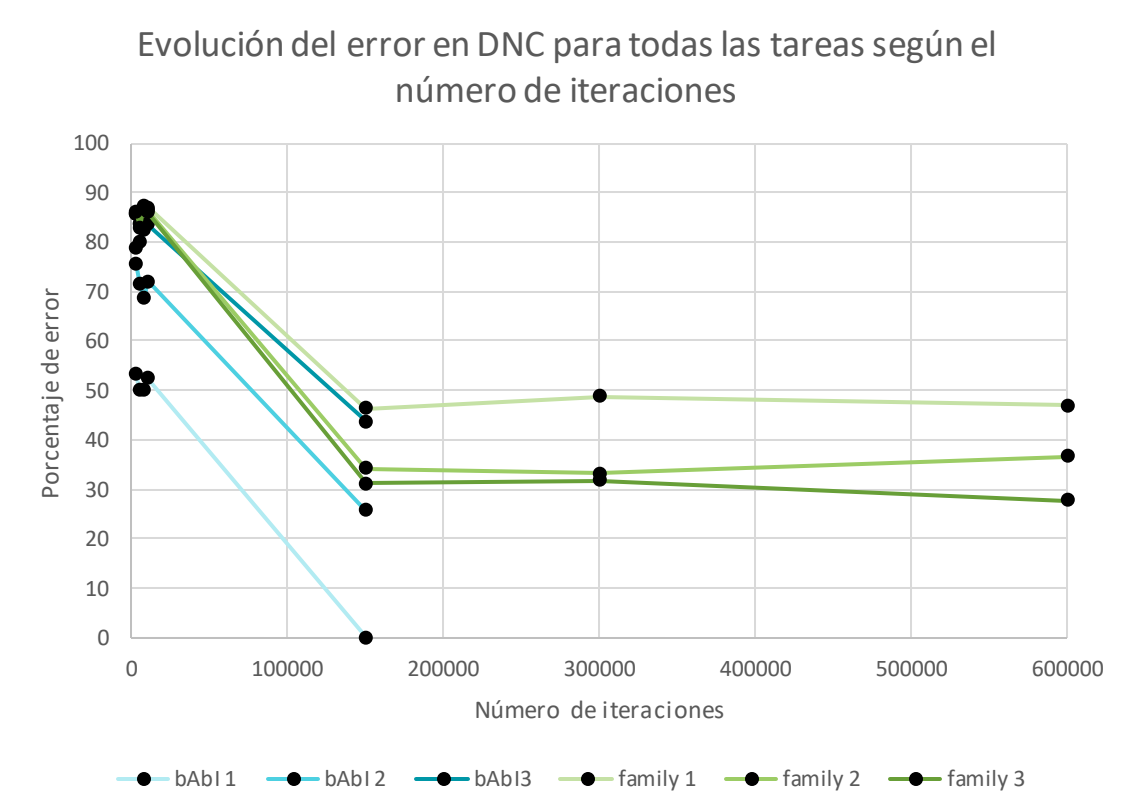

Ilustración 29: Evolución del error en DNC según el número de iteraciones

Para el dataset propio se ha entrenado este modelo con hasta 600.000 iteraciones, ya que se buscaban mejoras a largo plazo, aunque no se llegaron a encontrar. Con bAbI no se hicieron pruebas tan exhaustivas ya que la gráfica mostraba una tendencia más uniforme y dichas ejecuciones habrían requerido de más recursos.

Desgraciadamente, no se contaba con los medios necesarios para obtener más datos de este modelo. Hay que tener en cuenta que una sola ejecución de 600.000 iteraciones requería de más de 3 días para completar en entrenamiento y se hicieron 4 ejecuciones de estas, aparte de las ejecuciones más cortas. Sin embargo, con los resultados obtenidos ya podemos ver el comportamiento general de este modelo.

Como se puede apreciar en la **Ilustración 30**, la evolución del error a corto plazo es mínima, llegando a mostrar oscilaciones y ninguna mejora. Este gráfico se corresponde con las nubes de puntos a la izquierda del gráfico anterior.

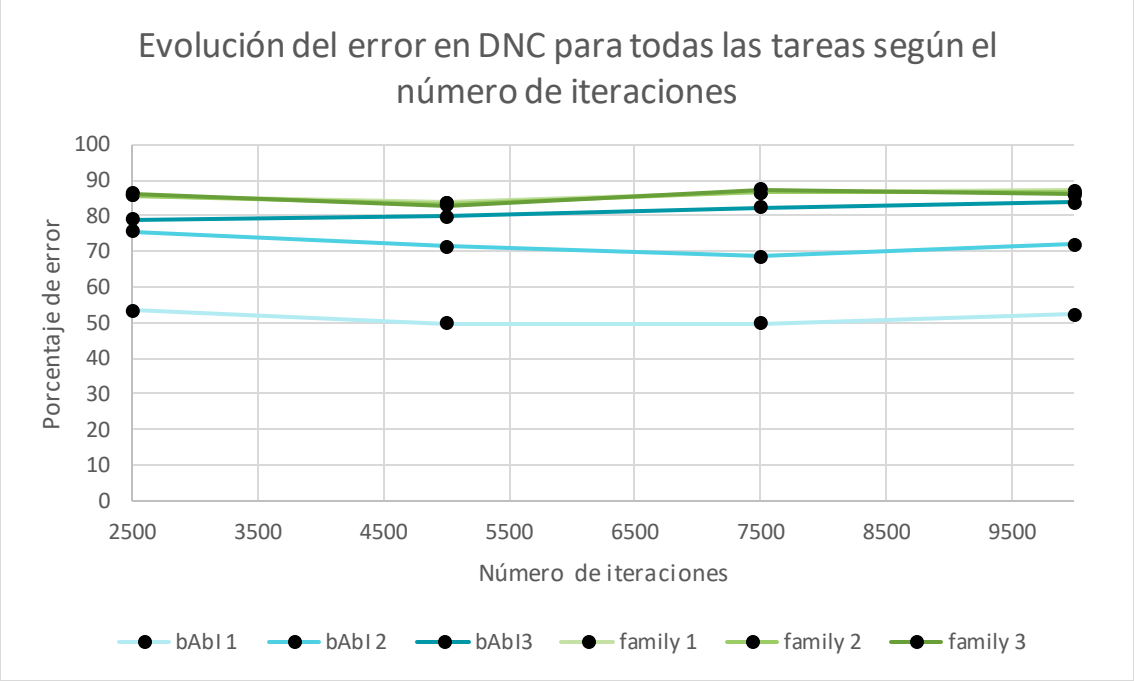

Ilustración 30: Evolución del error en DNC a corto plazo

Una desventaja de este modelo respecto a los demás, es el tiempo necesario para el entrenamiento y que los resultados obtenidos no han sido tan buenos como en el momento anterior, pero, por otro lado, es un modelo más general, que no está diseñado específicamente para las tareas utilizadas en este experimento, por lo que tiene un potencial mayor al poder aplicarse a otros tipos de tareas. En el entrenamiento de este modelo en la tarea del dataset propio usando 600.000 iteraciones, se necesitaron 3 días, 2 horas y 54 minutos.

6.4 PROPUESTA DE MEJORA

Debido a los resultados obtenidos y al comportamiento mostrado, se escogió DNC para proponer una mejora. Este modelo se muestra inmaduro al compararlo con los otros modelos, ya que le falta modularidad y carece de ciertas características aplicables más fácilmente a otros modelos más sencillos, como la transferencia de conocimiento.

La transferencia de conocimiento consiste en, teniendo una red para una tarea concreta, reentrenar la capa de salida para aplicarla a otra tarea. De esta forma, una gran parte del conocimiento adquirido se mantiene y solo se cambia la respuesta; haciendo que el entrenamiento sea mucho más corto y, potencialmente, la respuesta sea mejor.

Desde un punto de la adaptabilidad a diferentes problemáticas, este modelo parece tener la ventaja respecto a los otros dos modelos utilizados, pero, como hemos indicado anteriormente, es un modelo todavía inmaduro, pero con unas bases muy potentes.

7 CONCLUSIONES Y TRABAJO FUTURO

El desarrollo de este trabajo me ha permitido vivir en persona el proceso de experimentación e investigación. Para poder llevar a cabo este trabajo, he necesitado de una fase de aprendizaje previo bastante amplia que me ha brindado los conocimientos necesarios a partir de los obtenidos a lo largo del grado. En esta fase adquirí conocimientos sobre redes recurrentes y redes con memoria y sus características.

Por otro lado, a la hora de buscar proyectos de código abierto ya existentes que pudiera usar, descubrí la necesidad de crear código estructurado y documentado. Por ejemplo, el código de DNC de los creadores no era fácilmente reutilizable, pues usaba una librería no estándar y solo implementaba un ejemplo de tarea de copia.

Además, al tener que ajustar el código a un entorno determinado (Python 3.5 y Tensorflow 1.1), he tenido la oportunidad de valorar la retrocompatibilidad y la documentación oficial. Al carecer Tensorflow de estas características en las cantidades necesarias, el trabajo de adaptación de código requirió usar en algunos casos la técnica de prueba y error.

A la hora de llevar a cabo las ejecuciones de los modelos, tuvimos el problema del tiempo, pues se podía dar el caso de no tener los resultados disponibles incluso una semana después de empezar la ejecución. La necesidad de ejecutar varias veces con los mismos parámetros cada prueba para aportarle fiabilidad a los resultados hizo que se tuviera que reducir el abanico de pruebas que se querían realizar a aquellas que se consideraban más relevantes.

En cuanto a los resultados obtenidos, hemos visto resultados no esperados y resultados que se han hecho esperar. Estos son los casos de las redes LSTM donde añadir potencia a la red no siempre proporcionaba mejores resultados o de DNC, donde un entrenamiento podía necesitar de más de tres días.

Sin lugar a duda, el factor tiempo ha jugado en contra. Tener que adaptar el código, incluso en varias ocasiones, y el gran número de pruebas que se realizaron para obtener resultados, se llevaron la mayor parte del tiempo. En circunstancias ideales, con más recursos, se podrían haber llevado a cabo pruebas más exhaustivas. Y este sería el punto de partida de un trabajo futuro: a partir de los datos recogidos en este trabajo, se deberían buscar aquellos resultados menos esperados y aquellas pruebas no realizadas y realizar experimentos aún más concisos.

Sin embargo, se han conseguido realizar todas las tareas previstas en un principio. Con más o menos contratiempos, hemos podido replicar tres modelos y ver sus ventajas y desventajas ante diferentes ámbitos.

8 REFERENCIAS

Las imágenes y la explicación de las redes LSTM son adaptaciones de las mostradas en el artículo "Understanding LSTM Networks" (Olag, 2015).

Las imágenes y la explicación de la red MemN2N son adaptaciones de las mostradas en el paper "End-To-End Memory Networks" (Sukhbaatar, Szlam, Weston, & Fergus, 2015).

La imagen y la explicación de la red DNC son adaptaciones de las mostradas en el paper "Hybrid computing using a neural network with dynamic external memory" (Graves, y otros, 2016).

La **Ilustración 1** se obtuvo de http://ischlag.github.io/2016/06/04/how-to-use-tensorboard/, un tutorial de uso de Tensorboard.

9 ÍNDICE DE ILUSTRACIONES

10 BIBLIOGRAFÍA

All symbols in Tensorflow. (25 de Marzo de 2017). Obtenido de https://www.tensorflow.org/api_docs/python/

Graves, A., Wayne, G., Reynolds, M., Harley, T., Danihelka, I., Grabska-Barwińska, A., . . . Hassabis, D. (2016). Hybrid computing using a neural network with dynamic external memory. *Nature*, 472-476.

Guerra Artal, C. (9 de Marzo de 2017). *Some sample Tensorflow code.* Obtenido de Github: https://github.com/cayetanoguerra/TFSamples

Li, J., Miller, A., Chopra, S., Ranzato, M., & Weston, J. (1 de Marzo de 2017). *bAbI - Facebook Research.* Obtenido de Facebook Research: https://research.fb.com/downloads/babi/

Luna, D. (17 de Febrero de 2017). *End-To-End Memory Network using Tensorflow*. Obtenido de Github: https://github.com/domluna/memn2n

Olag, C. (Agosto de 2015). *Understanding LSTM Networks*. Obtenido de http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Python Software Foundation. (s.f.). *3.6.1 Documentation.* Obtenido de https://docs.python.org/3/

Samir, M. (14 de Enero de 2017). *A TensorFlow implementation of DeepMind's Differential Neural Computers (DNC)*. Obtenido de Github: https://github.com/Mostafa-Samir/DNC-tensorflow

Sukhbaatar, S., Szlam, A., Weston, J., & Fergus, R. (31 de Marzo de 2015). *End-To-End Memory Networks.* Obtenido de Cornell University Library: https://arxiv.org/abs/1503.08895