Desarrollo de Modelos de Deep Learning para comprensión de textos usando técnicas NLP

Trabajo Fin de Máster

Andrea Aguilar Ibáñez

Directores: Luis Mariano Esteban Escaño
Paula Peña Larena
Gerardo Sanz Sáiz

Universidad de Zaragoza
28 de noviembre de 2017

Prólogo

La Inteligencia Artificial surge a mediados del siglo XX, sobre todo con el trabajo desarrollado por el matemático Alan Turing, con la intención de dotar a las máquinas de la inteligencia propia del ser humano. La base de esta disciplina son la lógica matemática y la computación, que ha experimentado un gran avance debido a las mejoras en las tecnologías y de los ordenadores. Algunas de las líneas más importantes de la Inteligencia Artificial son los algoritmos genéticos y las redes neuronales artificiales, inspiradas en la idea de imitar la evolución biológica y el comportamiento de los cerebros biológicos, respectivamente. Estos últimos algoritmos son sobre los que se ha trabajado, en el área particular de la minería de textos, y el procesamiento del lenguaje natural (*Natural Language Processing, NLP*).

El objetivo del presente trabajo es estudiar el aunamiento de los sistemas basados en el conocimiento y los sistemas de almacenamiento. Surge de una propuesta de beca por parte del Instituto Tecnológico de Aragón (ITAINNOVA), de una duración de 9 meses.

Estos sistemas, conocidos como '*redes de memoria*', comienzan a aparecer en 2015 y son actualmente el estado del arte en modelos para la comprensión del lenguaje. La idea de estas redes es combinar algoritmos de aprendizaje automático, en particular las redes neuronales, con una memoria que permita almacenar y recuperar información de forma relevante según el objetivo buscado. Permiten realizar consultas utilizando frases con estructura sintáctica compleja, siendo esto una ventaja frente a los actuales buscadores de bases de datos que operan mediante palabras clave sin sintaxis.

Para evaluar el nivel de comprensión sobre información no estructurada, las redes son entrenadas a partir de textos y preguntas sobre los mismos, cuya respuesta es conocida, para así comparar la respuesta real con la respuesta que predice la red basándose en su propio "conocimiento".

Por tanto, el trabajo durante estos meses se ha centrado en el estudio, comprensión del comportamiento y limitaciones de dos extensiones de las redes de memoria, que son las *redes de memoria end-to-end* y las *redes de memoria clave-valor*, comparándolas con una red neuronal recurrente *LSTM*.

Se ha elegido el lenguaje de programación `Python` como principal herramienta debido a su gran versatilidad, ya que permite trabajar fácilmente con la librería de código abierto para aprendizaje automático, construcción y entrenamiento de redes neuronales desarrollada por `Google`, `TensorFlow`.

La realización de este Trabajo Fin de Máster no habría sido posible de no ser por la ayuda de Paula Peña, Rafael del Hoyo y el resto del personal de ITAINNOVA con el que he trabajado y convivido durante este periodo, sobre todo por mi compañera y gran amiga Rocío Aznar, que me animó a adentrarme en este *profundo* camino que es el Deep Learning. Mención especial también a Luis Mariano Esteban por sus consejos y dedicación, y a Gerardo Sanz. Por último, agradecer su paciencia y apoyo durante estos meses a mi familia y a Imanol.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

« Propongo que se considere la siguiente cuestión: "¿Las máquinas pueden pensar?". Para ello, lo primero sería dar definiciones del significado de los términos "máquina" y "pensar". »
— *Alan Turing*

Dentro de la Inteligencia Artificial se encuentra la rama denominada *Machine Learning* o aprendizaje automático, que otorga a las máquinas una capacidad de aprendizaje sin especificar explícitamente la manera en la que deben aprender y así poder crear sus propios patrones. Existen una serie de técnicas dentro de este campo centradas en la representación de la información mediante diferentes niveles, para así poder establecer relaciones más complejas entre los datos, agrupadas bajo el nombre de *Deep Learning* o aprendizaje profundo [1]. La utilización de métodos de Deep Learning, como por ejemplo las redes neuronales profundas (aquellas que cuentan con diferentes niveles de neuronas), es cada vez más frecuente cuando se trabaja con conjuntos de datos de gran tamaño, ya que permiten una mejor representación de los mismos y por lo tanto mejores resultados, en contraposición con los métodos clásicos de Machine Learning.

En los últimos años la tecnología está evolucionando vertiginosamente, lo que hace que aparezcan nuevas tendencias que producen una enorme cantidades de datos, como es la del *Big Data*, que está adquiriendo cada vez mayor notoriedad en las Tecnologías de la Información y la Comunicación (TICs.) Este concepto hace alusión a la gran cantidad de datos, tanto estructurados como no estructurados que, por su tamaño y heterogeneidad, plantean grandes dificultades para que puedan ser procesados por el software y los sistemas de gestión de bases de datos tradicionales.

Se crea la necesidad de desarrollar estrategias para abordar el tratamiento de esta ingente cantidad de datos, poder almacenarla y recuperar información relevante. Tareas que si los datos son no estructurados (por ejemplo, formato textual) conllevan una complejidad adicional para transformarlos a datos numéricos y trabajar con su consiguiente incremento en el volumen de datos. En este contexto, la Inteligencia Artificial, sirviéndose de la minería de textos, juega un papel importante.

En este capítulo se presenta una breve introducción a la minería de textos, una descripción del problema a resolver, los objetivos que se pretenden alcanzar, el contexto en el que se enmarca el trabajo y en el que se expone la motivación para la realización del mismo y por último, se detalla cómo se organiza el resto de la memoria.

1.1. Minería de textos

Se entiende por minería de textos la rama de la lingüística computacional cuyo objetivo es obtener información que no se encuentra de forma explícita en el conjunto de datos sobre el que se trabaja, que es un conjunto de textos. Una buena herramienta para tratar con estos datos no estructurados son las redes de Deep Learning. Dado que la extracción de esta información implícita en textos de gran tamaño es imposible de realizar por una persona en un tiempo razonable y abarcable, surge la necesidad de que las máquinas sean capaces de realizar estas tareas, y lo harán mediante la disciplina de la Inteligencia Artificial denominada como procesamiento del lenguaje natural, la cual se centra en la búsqueda, análisis y comprensión de mecanismos computacionales para la comunicación entre personas y máquinas mediante el uso del lenguaje humano (lenguaje natural), y así poder explotar toda la información disponible en el texto. Las diferentes etapas que comprende la minería de textos se muestran esquemáticamente en la figura 1.1 y se detallarán a continuación.

Algunos ejemplos donde estas técnicas son especialmente útiles son la extracción de información semántica que permita la recuperación de información relevante, el análisis de opiniones y sentimientos, la gestión de documentos y clasificación de los mismos, creación de chat-bots, traducción de textos, entre otros.

Etapas de la minería de textos

La recuperación de información relevante es la técnica de la minería de textos en la que se centran los modelos que se van a estudiar, con la aplicación de la 'búsqueda de respuestas' (*question answering, QA*). La búsqueda de respuestas consiste en que dada una cierta cantidad de documentos con datos en forma de texto, el sistema debe ser capaz de generar respuestas a preguntas planteadas en lenguaje natural, a partir de la información de la que dispone. Este sistema es bastante más complejo que otros sistemas de búsqueda de información como pueden ser las bases de datos. Puede verse como un problema de clasificación en el que hay tantas clases como posibles respuestas. Las respuestas a las preguntas serán las etiquetas para el entrenamiento.

Los datos que sirven para el entrenamiento de los modelos de Deep Learning en este caso están formados por una componente que son los datos de entrada, es decir el texto con la información, y las preguntas sobre el mismo, y otra componente que son los resultados deseados, es decir las respuestas a las preguntas planteadas, que son las etiquetas que permiten analizar las predicciones del modelo. Este es el formato necesario para tareas de aprendizaje supervisado, por tanto si no se dispone de un conjunto de datos en estas condiciones, tendría que generarse manualmente. La salida del modelo será una etiqueta de clase.

Cuando se cuenta con el conjunto de datos adecuado, debe hacerse un preprocesamiento del mismo para eliminar elementos innecesarios o hacer alguna transformación del texto, como por ejemplo que ninguna palabra empiece por mayúscula, o reemplazar un nombre compuesto por una clave para que se considere como una única palabra. Más adelante se explicará el procesamiento particular para cada conjunto de datos utilizado.

Una vez se dispone del texto en el formato deseado, debe transformarse a una representación numérica sobre la que ya pueden aplicarse los distintos modelos. En el caso que se está tratando, la información textual estará representada por matrices tridimensionales, las preguntas por matrices bidimensionales y las respuestas por vectores *one-hot*[1], respectivamente. Las representaciones matriciales se comentarán

[1] La codificación *one-hot* representa cada palabra del vocabulario total ordenado del texto, V, como un vector en $\{0,1\}^{|V|}$. Cada coordenada del vector corresponde a una palabra en V, por lo que el vector que representa a cada palabra tendrá un 1 en la posición correspondiente a dicha palabra y un 0 en el resto.

en el capítulo siguiente.

Teniendo todos los datos textuales en el formato numérico correspondiente pueden aplicarse los algoritmos de minería de datos pertinentes, en este caso los de redes neuronales recurrentes y redes de memoria *end-to-end* y *clave-valor*. Estos algoritmos se explicarán detalladamente en el capítulo 3.

Tras generar el modelo, debe validarse para comprobar su capacidad de aprendizaje y si es o no generalizable. Para ello debe contarse con un conjunto de validación que tenga el mismo formato que el de entrenamiento pero distinto contenido. Se utilizará como métrica de evaluación de clasificación multiclase la exactitud o *accuracy*, que indica la proporción de buenos clasificados por el modelo. Por ejemplo, si $\hat{a}$ =[[1, 0, 0, 0], [0, 0, 1, 0], [0, 1, 0, 0]] es la respuesta predicha (conjunto de respuestas) y a =[[1, 0, 0, 0], [0, 0, 0, 1], [0, 1, 0, 0]] es la respuesta correcta, el *accuracy* sería 2/3 ya que dos de las tres componentes predichas coinciden con la respuesta real.

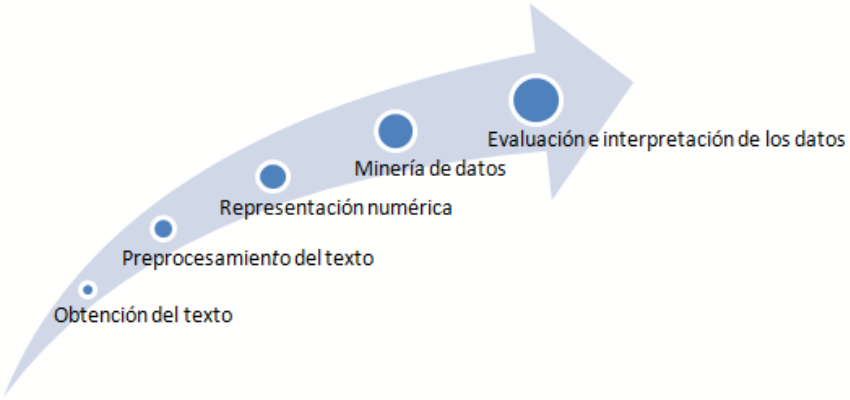


Figura 1.1: Esquema de la minería de textos.

1.2. Problema a resolver

En las últimas décadas las técnicas utilizadas para el procesamiento del lenguaje natural se centraban principalmente en algoritmos de aprendizaje automático o de regresión logística, pero desde hace unos años se han empezado a considerar las redes neuronales con aprendizaje profundo para estas tareas, ya que tienen una capacidad de aprendizaje y creación de nuevas variables que contienen información que podría haberse perdido en una creación de variables manual [2]. Dentro de las redes neuronales, las que interesan para esta tarea de aprendizaje son las redes neuronales recurrentes (*Recurrent Neural Networks, RNN*) [3], dado que por su configuración pueden almacenar información previa y posteriormente acceder y razonar sobre ella, lo que se asemeja a la posesión de una "memoria"[2]. Se estudiarán en profundidad en el capítulo 3.

Sin embargo, si se pretende tratar con una cantidad de información considerable, esta memoria con la que cuentan las redes neuronales recurrentes no es suficiente. Así pues se busca conseguir una capacidad de memoria más a largo plazo, introduciéndose a mediados de los 90 la variación de red neuronal recurrente *'Long Short-Term Memory' (LSTM)* [4], lo que se traduciría como memoria a largo y corto plazo. Este tipo de redes poseen, en lugar de nodos ocultos, unas celdas de memoria que permiten almacenar la información a corto y largo plazo, como su nombre indica. Están diseñadas para mejorar

[2]Entiéndase por memoria una componente de los modelos utilizada para acceder más rápido a la información y poder almacenar más cantidad de datos

las *RNN* estándar en cuanto a almacenamiento y acceso a la información se refiere. En ocasiones, la memoria de estas redes sigue sin ser suficiente para determinados conjuntos de datos, por lo que se intentan implementar nuevos modelos que mejoren a los anteriores. La extensión de redes neuronales que es más interesante en el caso que se está estudiando son las redes de memoria (*Memory Networks*) [5]. Desarrolladas por `Facebook research`, estas redes incorporan una componente de memoria a largo plazo en la que se puede leer y escribir, con el objetivo de poder realizar predicciones. Estos modelos se han aplicado en el contexto de búsqueda de respuestas, donde la memoria a largo plazo actúa como una base dinámica de conocimiento y la salida es una respuesta textual.

Como se verá en el capítulo 3, las redes neuronales recurrentes y las redes de memoria pueden combinarse, dando lugar a las redes neuronales de memoria (*Memory Neural Networks, MemNN*) [5]. Se estudiarán dos casos particulares de estas redes: las redes de memoria entrenadas *end-to-end* (*End-To-End Memory Networks, MemN2N*) [6] y las que poseen una memoria estructurada en forma *clave-valor* (*Key-Value Memory Networks, KV-MemNN*) [7]. Las primeras definen una memoria de gran tamaño, con diferentes ranuras, que puede codificar contexto tanto a corto como a largo plazo. Se realizan iteraciones sobre dicha memoria para buscar información relevante que permita responder a la pregunta planteada. En cada iteración, la información seleccionada como relevante se añade a la pregunta, creando una nueva característica, y así construir el contexto para la siguiente búsqueda de información relevante. En la última iteración se genera una respuesta de una lista de posibles candidatas, a partir de la combinación de todo el contexto seleccionado como relevante y la pregunta. Las segundas tienen un funcionamiento similar, con la particularidad de que cuentan con dos memorias distintas cuyas ranuras están asociadas por parejas de vectores $(k_1, v_1), \ldots, (k_M, v_M)$; lo que permite una búsqueda más sencilla de información relevante con respecto a la pregunta. Las memorias clave se diseñan con variables (características) de modo que se relacionen con la pregunta, mientras que las memorias valor debe diseñarse acorde a la respuesta.

En la figura 1.2 se muestra un esquema de todas las redes que se han nombrado y que van a estudiarse a lo largo del trabajo, en función de cómo se relacionan.

1.3. Objetivos

El objetivo principal de este Trabajo Fin de Máster consiste en el estudio de los recientes modelos de redes de memoria cuyo objetivo es afrontar el reto que presenta la comprensión de textos y posterior análisis por parte de las máquinas, todo mediante técnicas de minería de textos. Este estudio no pretende únicamente comprobar la adecuación de estos modelos al análisis y comprensión de información textual, sino que tiene como propósito la exploración más a fondo de los mismos, para comprender su funcionamiento y que permitan obtener mejores resultados. Esto ha precisado de una labor de análisis y optimización de los parámetros. A continuación se listan los objetivos específicos a alcanzar:

- ▷ Conocer a nivel teórico y práctico las herramientas y algoritmos de Machine Learning y en particular Deep Learning cuando se tienen grandes cantidades de datos.
- ▷ Diseñar un proceso que permita evaluar, categorizar y comprender un conjunto de textos, mediante técnicas de minería de textos.
- ▷ Entender las tecnologías Deep Learning y en especial los modelos de aprendizaje basados en técnicas recurrentes como son los modelos *LSTM*, o basados en redes de memoria entrenadas de extremo a extremo.
- ▷ Analizar la bibliografía existente en lo referente a '*question answering*'.

- ▷ Diseñar experimentos y metodologías que permitan evaluar el rendimiento de estos algoritmos a nivel estadístico en esa temática.
- ▷ Programar e integrar los modelos anteriormente descritos mediante las herramientas de análisis de datos.
- ▷ Modificar y optimizar los modelos para obtención de mejores resultados.

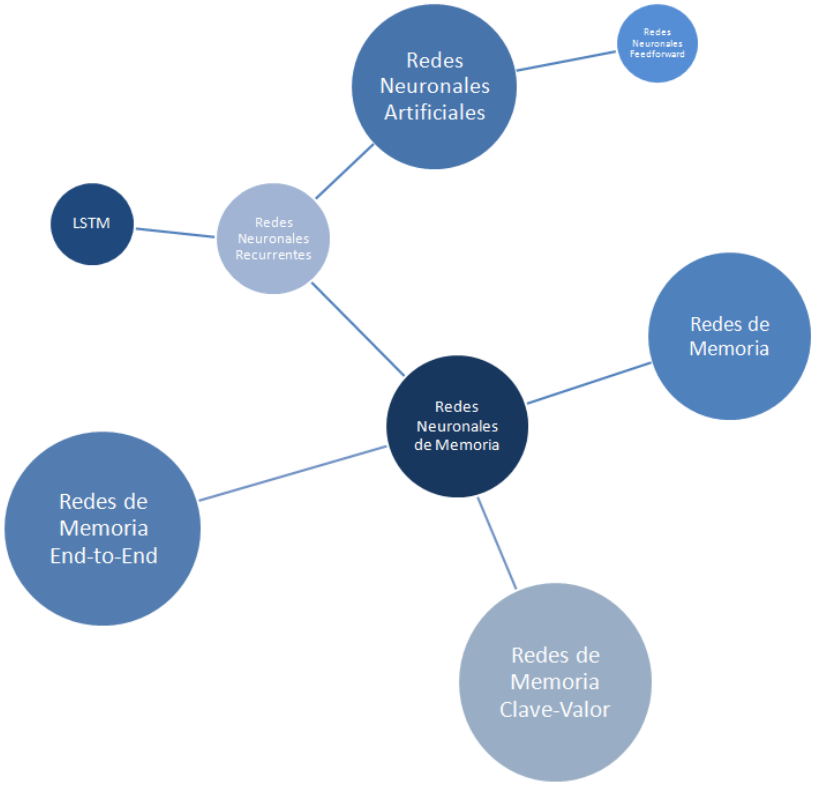


Figura 1.2: Esquema de redes y relaciones.

1.4. Contexto y motivación

La realización de este trabajo se enmarca dentro de las líneas de trabajo del Grupo de Big Data y Sistemas Cognitivos del Instituto Tecnológico de Aragón (ITAINNOVA). Dentro de este grupo y desde hace varios años, se está investigando y trabajando muy activamente en los ámbitos de la Inteligencia Artificial, Big Data, Web Semántica y Sistemas de la Información, a través de su participación en proyectos de diversa índole y procedencia. Este trabajo en particular, se ha centrado en la utilización de tecnologías de la Inteligencia Artificial, Big Data y métodos para la optimización de resultados.

La motivación principal para la realización de este Trabajo Fin de Máster se fundamenta en la necesidad de aportar soluciones al ámbito de *question answering* o búsqueda de respuestas a partir de textos, lo que requiere una mejora y optimización en los métodos de búsqueda semántica. Además, como motivación a nivel personal estaba el afrontar un problema real con el que pueden encontrarse las empresas dedicadas a trabajar con proyectos de Big Data, además de poder trabajar en un entorno multidisciplinar y tener un primer contacto con el mundo laboral.

Para este trabajo se van a tener que aunar distintos conocimientos adquiridos en el máster: modelado de problemas, minería de datos y programación. También profundizar personalmente en otras disciplinas tales como la minería de textos y el Deep Learning.

1.5. Estructura de la memoria

La presente memoria pretende exponer, de forma precisa y descriptiva, el trabajo desarrollado a lo largo de las distintas fases y etapas seguidas en el desarrollo del trabajo, así como los resultados obtenidos y las conclusiones finales. La memoria se estructura en una parte central de 5 capítulos: el primero se ha dedicado a introducir la temática del trabajo, el segundo a la representación de la información, el siguiente a los modelos de redes neuronales y redes de memoria, otro centrado únicamente al análisis y optimización de los parámetros de estos modelos, y el último que resume las conclusiones obtenidas, posibles trabajos futuros y aplicaciones. Al final se adjuntan dos anexos que complementan la memoria.

El capítulo 2 se centra en la representación de la información disponible. Es muy importante que los conjuntos de datos estén codificados de una manera adecuada, por lo que el trabajo de preprocesado de los textos es clave para un correcto entrenamiento de las redes neuronales. Se introducen diferentes métodos de representación generales y posteriormente las representaciones particulares en función del conjunto de datos con el que se vaya a trabajar, con algunos ejemplos, además de una explicación sobre estos conjuntos y su estructura.

En el capítulo 3 se introducen las redes neuronales artificiales, en particular las redes neuronales recurrentes, y diferentes algoritmos basados en el gradiente y la minimización de los errores de predicción, utilizados para el aprendizaje durante el entrenamiento de estas redes. Después se explican dos estructuras distintas de red neuronal recurrente, ya que serán las que se utilicen en uno de los tres modelos entrenados. Además de utilizarse como modelo independiente de estudio, aparecerán incluidas dentro de las redes de memoria. Tras esto, el capítulo explica los modelos que son principal objeto de estudio del Trabajo Fin de Máster, las redes de memoria. Se enumeran los diferentes parámetros presentes en estos modelos y que se variarán para intentar encontrar entrenamientos óptimos. Por último, se profundiza en los casos particulares de estos modelos que cuentan con una red neuronal recurrente en alguna de sus componentes: redes de memoria *end-to-end* y clave-valor, así como posibles modificaciones y unos ejemplos sencillos que ayuden a visualizar su funcionamiento.

En el capítulo 4 se expone la parte práctica del Trabajo Fin de Máster: un análisis en función de diferentes parámetros de la optimización de los tres modelos y una posterior comparación entre ellos. El propósito de este estudio es analizar cómo la variación de los parámetros influye considerablemente en los resultados obtenidos. Asimismo, se lleva a cabo otro estudio sobre la influencia del tipo de formato en que se presenta la información. Para visualizar el entrenamiento de las redes se representará el ajuste de los resultados mediante gráficas creadas con el lenguaje de programación `R` [8].

Por último, en el capítulo 5 se expondrán las conclusiones obtenidas a raíz de los objetivos planteados, posibles mejoras y líneas futuras de trabajo, además de detallar las posibles aplicaciones.

En los anexos se incluye parte del código de programación desarrollado para este trabajo. En el anexo A se adjuntan las funciones principales implementadas en `Python` para el funcionamiento de las redes de memoria. El anexo B incluye el código en `R` que ha sido necesario para la creación de las gráficas para la comparación de resultados.

Capítulo 2

Representación de la información

« La clave de la inteligencia artificial siempre ha sido la representación. »
— Jeff Hawkins

En minería de textos, las etapas de preprocesamiento y transformación del conjunto de datos son clave para un correcto funcionamiento del modelo que vaya a aplicarse. Tanto la estructuración del texto en el formato que interese como la generación de un conjunto de datos adecuado, así como el preprocesamiento del mismo y la posterior representación en variables numéricas, requiere un trabajo de programación informática. Una parte importante del tiempo precisado para el desarrollo de este Trabajo Fin de Máster ha sido dedicado a la adaptación de las librerías para un correcto procesamiento y representación de los datos. En particular, se han modificado y ampliado las librerías disponibles inicialmente para que la información pudiera almacenarse y trabajarse en dos memorias distintas para el funcionamiento del modelo clave-valor, y también para que la información sobre uno de los conjuntos de datos pudiera procesarse para tres tipos distintos de formatos de representación.

Principalmente se ha trabajado con el lenguaje de programación `Python` [9], tanto por adecuación al desarrollo de posteriores proyectos de la empresa como por la numerosa documentación, librerías y trabajo existente sobre redes neuronales en este lenguaje. Como librerías de `Python` utilizadas a destacar están `TensorFlow` [10], que tiene implementadas numerosas funciones útiles para redes neuronales, y `Keras` [11], que es una API de redes neuronales de alto nivel, es decir una librería útil para Deep Learning.

También se ha precisado aprender y trabajar con `Lua` [12] para la simulación de las *tareas bAbI* [13] [14] (explicadas en la sección 2.2). `Lua` es un lenguaje de programación imperativo, estructurado y bastante ligero que fue diseñado como un lenguaje interpretado con una semántica extensible.

En este segundo capítulo se presentan los dos conjuntos de datos con los que se ha trabajado: *tareas bAbI* y *WikiMovies*, ambos desarrollados por el equipo de investigación sobre Inteligencia Artificial de Facebook, dentro del proyecto *bAbI*[1], el cual está formado por un conjunto de recursos cuyo objetivo es comprender y razonar de forma automática a partir de textos [15]. También se comentan las diferentes formas de representación en las que pueden encontrarse los datos textuales, así como la manera de procesarlos y representarlos numéricamente para después codificarlos. Todos los ejemplos que se han incluido en este capítulo están escritos en inglés, ya que es el idioma original en el que se presentan estos conjuntos de datos.

[1] El proyecto *bAbI* está formado por otros conjuntos de datos y estudios además de estos.

2.1. Representación característica e inmersión de palabras

Dado que se trata con una gran cantidad de datos, si se utilizase siempre la representación *one-hot* introducida en el capítulo anterior se tendrían vectores muy dispersos y de gran dimensión (una dimensión por cada característica[2]), siendo esto un inconveniente a la hora de trabajar con redes neuronales. Por tanto conviene hacer un trabajo previo para reducir estas dimensiones y así poder operar con vectores más densos.

Se transforman los datos de entrada tal cual se reciben (frases de los textos) a la representación característica en un espacio de dimensión d, representados por un vector en dicho espacio. Estos vectores serán parámetros a entrenar por el modelo. Esta transformación recibe el nombre de inmersión o *embedding*. Una inmersión es una aplicación de objetos discretos, como pueden ser las palabras, a vectores de números reales. Es decir, cada palabra tiene como imagen un vector del espacio de inmersión. Esta representación permite que vectores similares representen características que tengan algún tipo de correlación. Sin embargo, si no existe correlación alguna, entonces es conveniente utilizar la representación *one-hot*, como es el caso de las respuestas a las preguntas sobre los de datos, pues cada respuesta es diferente de las demás.

2.1.1. Bolsa de palabras

El modelo de bolsa de palabras (*Bag-of-words, BOW*) es un método para crear un diccionario de todas las palabras presentes en un texto pero ignorando el orden de las mismas dentro de la frase y del propio texto. Una frase se representará mediante un vector cuyas componentes serán el número de veces que aparece cada palabra en el diccionario, en la posición que corresponda.

Ejemplo: A partir de un fragmento de texto se ha creado el siguiente diccionario:

```
[an, and, apple, bedroom, drops, into, kitchen, Sam, take, the, then, to, walks]
```

entonces la frase "*Sam walks into the bedroom and then to the kitchen*" quedará representada por el vector [0, 1, 0, 1, 0, 1, 1, 1, 0, 2, 1, 1, 1].

Dado $x_i = x_{i1}, \ldots, x_{in}$, siendo x_{ij} cada una de las palabras que conforman la frase x_i y A la matriz de inmersión de dimensiones $d \times V$, la representación característica de x_i será:

$$y_i = \sum_j A x_{ij}.$$

2.1.2. Codificación de posición

En algunas ocasiones la representación anterior no es suficiente debido a que es necesario tener en cuenta la posición de la palabra en la frase, ya que puede modificar el significado.

Ejemplo: Las preguntas "*What is north of the bedroom?*" y " *What is the bedroom north of?*" tienen el mismo número de palabras pero en diferente orden, y es evidente que requieren respuestas diferentes.

Para esto, siendo $x_i = x_{i1}, \ldots, x_{in}$, con x_{ij} cada una de las palabras que conforman la frase x_i, para $i = 1, \ldots, m$ con m el número de frases que forman el texto, y A la matriz de inmersión de dimensiones $d \times V$, la representación característica de x_i será:

[2]Entiéndase por característica una componente lingüística específica, como pueden ser las palabras, etiquetas gramaticales o cualquier otra información lingüística. Son las variables del modelo.

$$y_i = \sum_j l_j \cdot Ax_{ij},$$

donde '$\cdot$' es una multiplicación elemento a elemento y l_j es un vector columna que tiene la estructura

$$l_{kj} = 1 - \frac{j}{J} - \frac{k}{d}\left(1 - \frac{2j}{J}\right),$$

para $j = 1, \ldots, J$, siendo J el número de palabras en la frase x_i y k la posición de los elementos en el vector columna [6].

2.1.3. Tensores

Como se ha indicado previamente, `TensorFlow` tiene implementadas funciones útiles para redes neuronales, por lo que es la librería principal utilizada en las implementaciones de los modelos. Como su nombre indica, esta herramienta permite definir y ejecutar cálculos que involucran *tensores*, por lo que se ha utilizado a la hora de definir las variables del modelo y los vectores donde se almacenan los datos de entrada. Un tensor es un elemento matemático que generaliza los conceptos de escalar, vector y operador lineal de una manera que sea independiente de cualquier sistema de referencia elegido. Pueden verse como matrices n-dimensionales[3]. Un escalar es un tensor de orden cero, un vector es un tensor de primer orden, una matriz un tensor de segundo orden, y así sucesivamente.

Por ejemplo, como se explicará a continuación, para el conjunto de datos *tareas bAbI* las preguntas sobre el texto se almacenan en tensores de orden 2 (matrices) y las frases que aportan la información en tensores de orden 3 (cubos de números). En la figura 2.1 se ilustra cómo sería este almacenamiento, aunque con las palabras del texto y no con los números correspondientes tras el procesamiento, para una fácil visualización.

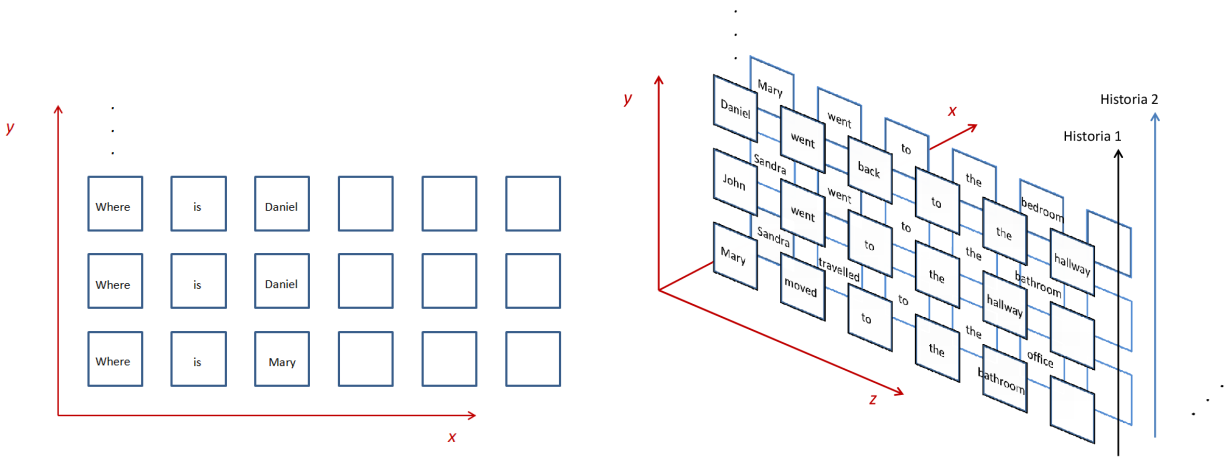


Figura 2.1: Almacenamiento de la información en tensores.

Una vez explicados estos métodos generales de representación y codificación de información textual, se van a describir en las secciones siguientes los dos conjuntos de datos utilizados. Tras esto, en la sección 2.4, se detallarán las representaciones y codificaciones particulares para cada conjunto.

[3] Más información sobre tensores en `https://www.tensorflow.org/programmers_guide/tensors`

2.2. Tareas bAbI

Este conjunto de datos fue creado mediante simulación con la intención de medir el nivel de comprensión del lenguaje de cualquier sistema mediante la búsqueda de respuestas. En realidad está formado por 20 conjuntos de datos distintos, las denominadas *tareas bAbI*, y cada uno de ellos representa un nivel de comprensión y razonamiento diferente: concatenación de hechos, inducción, deducción y más [13]. Cada tarea está formada por diferentes historias enumeradas, y cada vez que los modelos procesan una nueva línea de dicha historia, actualizan el conocimiento en la componente de memoria. Esto justifica la forma de procesar el texto que se explicará más adelante, donde cada vez que se recibe información nueva, esta se almacena conjuntamente con la información que ya se tenía, en lugar de almacenarla por separado.

Cada una de las tareas consta de 1000 preguntas para el entrenamiento (*train*) y 1000 para la evaluación (*test*), con sus correspondientes historias asociadas. A continuación se presentan los tipos de tareas y un pequeño fragmento de cada una:

Tarea 1 Está formada por preguntas precedidas por diferentes frases pero la respuesta solo puede deducirse de una de ellas. Es la situación más sencilla.

Daniel went back to the hallway.
Sandra moved to the garden.
Where is Daniel? hallway

Tarea 2 Va un paso más allá que la tarea anterior, pues para responder a la pregunta es necesario contar con la información de dos frases.

Mary got the football there.
John went to the kitchen.
Mary went back to the garden.
Where is the football? garden

Tarea 3 Es el mismo caso que la tarea 2, pero presenta algo más de complejidad que la anterior al tener que combinar tres frases para poder responder a la pregunta.

Sandra picked up the milk.
Sandra travelled to the hallway.
John journeyed to the hallway.
Sandra went to the kitchen.
Where was the milk before the kitchen? hallway

Tarea 4 Tanto para responder a las preguntas de esta tarea como de la siguiente, es indispensable poder diferenciar y reconocer sujetos y objetos.

The bathroom is east of the bedroom.
The kitchen is west of the bedroom.
What is the bedroom east of? kitchen

Tarea 5 Se necesitan diferenciar tres argumentos y se responden preguntas sobre quién da un objeto, quién lo recibe y cuál es el objeto en cuestión.

Bill went to the bedroom.	
Bill gave the football to Fred.	
Fred handed the football to Jeff.	
What did Bill give to Fred?	football
Who received the football?	Jeff

Tarea 6 Esta tarea prueba la habilidad de responder preguntas del tipo verdadero/falso, teniendo que responder 'sí o no' a partir sólo de una frase soporte.

Mary moved to the bathroom.	
Sandra journeyed to the bedroom.	
Is Sandra in the hallway?	no
Mary went back to the bedroom.	
Daniel went back to the hallway.	
Is Daniel in the bathroom?	no

Tarea 7 Prueba la capacidad de realizar operaciones de conteo preguntando por el número de objetos con una cierta propiedad, tales como ser llevado por un sujeto determinado.

Mary moved to the bathroom.	
Mary took the football there.	
Mary went to the hallway.	
How many objects is Mary carrying?	one
Mary dropped the football.	
How many objects is Mary carrying?	none

Tarea 8 De manera similar a la anterior tarea, esta prueba la capacidad de generar una respuesta en forma de lista. Esto puede verse como operaciones básicas de búsqueda de información de bases de datos.

Daniel picked up the football there.	
Daniel dropped the football there.	
Mary grabbed the football there.	
What is Daniel carrying?	nothing
Mary travelled to the office.	
Mary took the apple there.	
What is Mary carrying?	football,apple

Tarea 9 Esta tarea incluye frases soporte con información falsa (con negación). Tiene como requisito la tarea 6 ya que se basa en el formato de respuesta 'sí o no'.

John travelled to the kitchen.	
Mary is in the hallway.	
Daniel is no longer in the garden.	
Is John in the kitchen?	yes
Is Daniel in the garden?	no

Tarea 10 En esta tarea se consideran frases que describen posibilidades en lugar de certezas. Ahora las posibles respuestas son 'sí, no y quizás'.

Fred is in the office.	
Bill is either in the office or the kitchen.	
Is Fred in the bedroom?	no
Is Bill in the office?	maybe

Tarea 11 Refleja la correferencia más simple. Es decir, cuando se hace referencia al mismo sujeto mediante dos o más expresiones lingüísticas en el mismo texto. Por ejemplo en el siguiente fragmento se hace primero referencia a una persona por medio de su nombre (Mary) y en la siguiente oración con el pronombre personal 'ella'.

Mary went back to the bathroom.	
After that she went to the bedroom.	
Where is Mary?	bedroom

Tarea 12 Esta tarea considera sujetos múltiples que realizan acciones simultáneamente. Se analiza así la capacidad de diferenciación de sujetos.

Sandra and Daniel moved to the kitchen.	
Sandra and John moved to the garden.	
Where is Sandra?	garden

Tarea 13 Prueba la correferencia igual que la tarea 11, pero ahora el pronombre puede referirse a más de un sujeto, como en la tarea 12.

Daniel and Sandra journeyed to the office.	
Then they moved to the bedroom.	
Mary and Daniel travelled to the bathroom.	
After that they moved to the hallway.	
Where is Daniel?	hallway

Tarea 14 Esta tarea evalúa el entendimiento de uso de expresiones temporales, preguntando por el orden en que suceden las diferentes acciones.

Julie went to the school this morning.	
Yesterday Julie went to the office.	
Julie went to the cinema this evening.	
Where was Julie before the school?	office
Where did Julie go after the school?	cinema

Tarea 15 Aquí se analiza la deducción básica a través de propiedades heredadas.

Mice are afraid of wolves.	
Cats are afraid of sheep.	
Gertrude is a mouse.	
What is gertrude afraid of?	wolf

Tarea 16 Prueba la inducción básica a través de propiedades heredadas.

Julius is a swan.
Julius is green.
Greg is a swan.
What color is Greg? green

Tarea 17 Esta tarea se centra en evaluar el razonamiento espacial preguntando por las posiciones relativas de diferentes objetos.

The triangle is above the pink rectangle.
The blue square is to the left of the triangle.
Is the pink rectangle to the right of the blue square? yes
Is the blue square below the pink rectangle? no

Tarea 18 El objetivo aquí es evaluar el razonamiento sobre el tamaño de unos objetos con respecto a otros.

The box of chocolates fits inside the chest.
The box is bigger than the chest.
Does the box fit in the box of chocolates? no
Is the box bigger than the box of chocolates? yes

Tarea 19 Esta nueva tarea está enfocada a encontrar el camino entre diferentes localizaciones, esto es dadas las descripciones de dichas localizaciones, las preguntas se refieren a cómo se llegaría de un lugar a otro.

The office is east of the hallway.
The kitchen is north of the office.
The garden is west of the bedroom.
The office is west of the garden.
The bathroom is north of the garden.
How do you go from the kitchen to the garden? south, east

Tarea 20 La última tarea se orienta a conocer por qué un sujeto realiza una determinada acción, dado el estado en que se encuentran. Tiene que aprender la acción que implica el estado del sujeto.

Sumit went back to the bedroom.
Jason is thirsty.
Why did sumit go to the bedroom? tired
Where will Jason go? kitchen

La búsqueda de respuestas a partir de este conjunto de datos es un problema complejo, sobre todo para ciertas tareas, ya que como se ha explicado en cada tarea, se requiere una capacidad de diferenciar sujetos, lugares, objetos, etc., y razonar sobre ellos. La dificultad de entrenamiento de cada tarea tendrá sus consecuencias en las evaluaciones del modelo.

2.3. WikiMovies

El conjunto de datos *WikiMovies* está formado por información sobre un gran número de películas junto con preguntas y sus respectivas respuestas. La información está disponible en tres formatos distintos: texto completo, base de conocimiento y conocimiento extraído (esta última representación es algo intermedio entre las otras dos (véase sección 4.1 del artículo original [7] para más detalle). Se creó con el objetivo de disponer de un conjunto de datos lo suficientemente grande para aplicar técnicas de aprendizaje automático y poder analizar fácilmente la capacidad de comprensión de los modelos. Dado que este Trabajo Fin de Máster se centra en el análisis y comprensión de textos, se ha trabajado sobre el conjunto de datos con texto completo.

Existen 13 tipos de preguntas en función de las relaciones entre los elementos de la base de conocimiento: actor-película, director-película, película-actores, película-director, película-género, película-clasificación IMBD, película-votos IMBD, película-idioma, película-etiquetas, película-escritor, película-año, etiqueta-película, escritor-película.

El documento original cuenta con 22999 películas distintas (separadas en 92983 líneas), el conjunto de preguntas para el entrenamiento es de 196453, el de validación de 10000 preguntas y el de test de 10000 [4]. Además se cuenta también con un documento que almacena todas las entidades presentes en los conjuntos anteriores, habiendo un total de 75542. Para este conjunto de datos se entiende por *entidad* cualquier nombre propio, año, localización, idioma u otro sustantivo que aparezca en el texto y pueda responder a las preguntas que vayan a realizarse.

A continuación se presenta un fragmento del conjunto de datos, que refleja la información disponible para una película, y también una selección del tipo de preguntas presentes en el conjunto de entrenamiento.

Información

Kismet (1944 film): Kismet is a 1944 Metro-Goldwyn-Mayer film in Technicolor starring Ronald Colman, Marlene Dietrich, Joy Page, and Florence Bates. James Craig played the young Caliph of Baghdad, and Edward Arnold was the treacherous Grand Vizier. It was directed by William Dieterle, but was not a success at the box office. The film is based on the play of the same name by Edward Knoblock, which was also the basis for a 1953 musical. The play had been filmed three times before, in 1914, 1920, and again in 1930 by Warner Brothers in an English version directed by John Francis Dillon and in a German-language version directed by William Dieterle.

Preguntas

what movies did Edward Knoblock write?	Kismet, The Sin of Madelon Claudet, Chu Chin Chow
what year was Kismet released?	1944
which person wrote Kismet?	John Meehan, Edward Knoblock
what is the language spoken in the film Kismet?	English
what genre of movie is Kismet?	Adventure, Fantasy

[4] La información disponible sobre las películas formará parte tanto del conjunto de entrenamiento, como de validación y test, en adición de las preguntas consideradas en cada caso.

2.4. Procesamiento del texto y codificación

2.4.1. Tareas bAbI

Considérese una de las historias presentes en el fichero de texto de entrenamiento correspondiente a la tarea 1, mostrada en la figura 2.2. Una historia esta formada por diferentes subhistorias, formadas por todas las frases previas a cada pregunta (sin contar con las preguntas anteriores). Una pregunta puede ser respondida por la frase inmediatamente anterior, o por la primera frase de la historia.

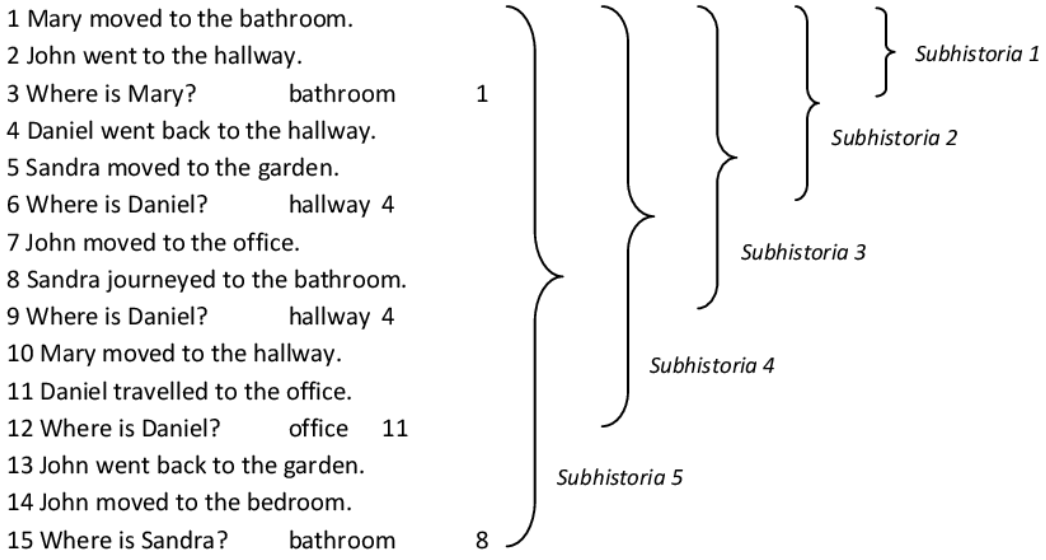


Figura 2.2: Historia dentro de una tarea.

El conjunto de datos que se considera como ejemplo para explicar el procesamiento contiene dos historias más además de esta, luego el conjunto de entrenamiento consta de 15 preguntas con sus correspondientes 15 subhistorias, formando un total de 45 líneas. El conjunto de evaluación tendrá el mismo tamaño, el mismo número de historias y preguntas.

Se guardan en un vector las frases de cada subhistoria seguidas de su correspondiente pregunta con la respuesta asociada. A partir de esto se crea un diccionario formado por todas las palabras presentes en el conjunto de datos completo y se asigna un índice a cada palabra, guardando el índice cero para el símbolo nulo:

```
[ ( [ ['mary', 'moved', 'to', 'the', 'bathroom'],
      ['john', 'went', 'to', 'the', 'hallway']   ],
    ['where', 'is', 'mary'], ['bathroom']          ),

  ( [ ['mary', 'moved', 'to', 'the', 'bathroom'],
      ['john', 'went', 'to', 'the', 'hallway'],
      ['daniel', 'went', 'back', 'to', 'the', 'hallway'],
      ['sandra', 'moved', 'to', 'the', 'garden']         ],
    ['where', 'is', 'daniel'], ['hallway']                 ),

                          .  .  .
                                                          ]
```

```
DICCIONARIO:
{'hallway': 6, 'bathroom': 2, 'garden': 5, 'journeyed': 9, 'office': 13, 'is': 7,
'sandra': 14, 'moved': 12, 'back': 1, 'went': 18, 'to': 16, 'daniel': 4, 'bedroom'
: 3, 'the': 15, 'john': 8, 'where': 19, 'mary': 11, 'travelled': 17, 'kitchen': 10}
```

A continuación se sustituyen las palabras por su índice y se separan en frases, preguntas y respuestas. Las respuestas se codifican como vectores *one-hot*. Para el conjunto de datos considerado, las dimensiones de los tensores en los que se almacenan las historias, las preguntas y las respuestas serán (15, 10, 6), (15, 6) y (15, 20) respectivamente. En el caso de las historias, la primera dimensión vendrá dada por el número de subhistorias (que es igual al número de preguntas) presentes en el conjunto, la segunda es la longitud de la historia más larga, es decir hay una historia que está formada por 10 frases. La tercera componente representa la longitud de la frase más larga (considerando tanto frase de historia como frase de pregunta), es decir la frase con más número de palabras. Las dimensiones de los tensores que almacenan las preguntas quedan determinados de manera similar por el número de preguntas y la longitud de la frase más larga, y las respuestas se almacenarán en un tensor que será un vector de vectores *one-hot* con dimensiones dadas por el número de respuestas (15 pues hay una respuesta por cada pregunta) y el número de palabras en el diccionario.

Dado que `TensorFlow` no puede trabajar con matrices irregulares, todas las frases del conjunto de datos tendrán que tener la misma longitud, luego para aquellas cuya longitud sea menor que la máxima, su vector numérico se rellenará con ceros hasta llegar a la máxima longitud. Lo mismo si alguna de las historias tiene longitud menor de la máxima establecida: tendrán que añadirse vectores de ceros de longitud 6 (longitud de frase) hasta completar el tamaño de la memoria.

Así, las primeras frases, preguntas y respuestas del ejemplo mostrado en la figura 2.2 quedarían codificadas de la siguiente manera:

```
[ [ [ 8 18 16 15  6  0]           john went to the hallway
    [11 12 16 15  2  0]           mary moved to the bathroom
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0] ]

  [ [14 12 16 15  5  0]           sandra moved to the garden
    [ 4 18  1 16 15  6]           daniel went back to the hallway
    [ 8 18 16 15  6  0]           john went to the hallway
    [11 12 16 15  2  0]           mary moved to the bathroom
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0]
    [ 0  0  0  0  0  0] ]
        .  .  .
                        ]
```

Es importante remarcar que las frases se indexan en orden inverso, por lo que los primeros vectores de cada grupo de vectores corresponden con la última frase de la subhistoria.

```
[ [19  7 11  0  0  0]           where is mary
  [19  7  4  0  0  0]           where is daniel
  [19  7  4  0  0  0]           where is daniel
  [19  7  4  0  0  0]           where is daniel
  [19  7 14  0  0  0]
  [19  7 14  0  0  0]
  [19  7 14  0  0  0]
  [19  7 14  0  0  0]
  [19  7  8  0  0  0]
  [19  7  4  0  0  0]
  [19  7  8  0  0  0]
  [19  7 11  0  0  0]
  [19  7  8  0  0  0]
  [19  7  8  0  0  0]
  [19  7 14  0  0  0] ]
```

```
[ [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]       bathroom
  [ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0]       hallway
  [ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0]       hallway
  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0]       office
  [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]       bathroom
  [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
  [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
  [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
  [ 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0]
  [ 0  0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0]
  [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
  [ 0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
  [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
  [ 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0]
  [ 0  0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0] ]
```

Una vez se tienen los conjuntos de entrenamiento y test vectorizados, el de entrenamiento se divide aleatoriamente por validación cruzada, considerando un 90 % de los datos como conjunto de entrenamiento y un 10 % de validación. El primer conjunto se utiliza para entrenar el modelo y el segundo para elegir el más adecuado. Después se aplicará el modelo sobre el conjunto test para obtener su *accuracy* y probar si es generalizable a otros datos.

2.4.2. WikiMovies

En este caso el procedimiento de almacenar y transformar la información disponible sobre películas y las preguntas referentes a ellas es bastante similar. La información sobre las películas tiene diferentes representaciones, pero se destacan tres: nivel frase, nivel ventana y nivel tripleta. En los tres casos se guarda la información en tensores de orden 3, y las preguntas en tensores de orden 2. Además, dependiendo del modelo que se entrene sobre este conjunto el almacenamiento será diferente: para el modelo

MemN2N la información se guardará en la memoria tal cual se presenta, ya sea como frase, como ventana o como tripleta; sin embargo para el modelo *KV-MemNN* debe separarse en una parte referente a la clave y otra asociada al valor. A continuación se detallarán las representaciones en el caso de considerar el modelo clave-valor. En el anexo A se muestra el código principal de este modelo y las diferentes funciones que ha sido necesario implementar para procesar la información y para su correcto funcionamiento.

El conjunto de datos que se considera como ejemplo para explicar el procesamiento contiene información sobre 5 películas y 49 preguntas.

Formato frase

Este formato es el que presenta la información separando el texto disponible en frases. Cada película se almacena en un vector compuesto por vectores que contienen cada frase con información sobre dicha película. En este caso, ambas memorias guardarán la frase completa, no hay diferencias entre clave y valor. Este caso particular es idéntico al modelo *MemN2N*. La dimensión del tensor de frases para el entrenamiento será inicialmente (5, 6, 66), dado que se tiene información sobre 5 películas, la película que más frase tiene de información es 6 y la longitud máxima de frases es 66 (considerando también las preguntas). Puesto que las componentes x y z deben coincidir con las x e y de las preguntas, debe redimensionarse a un tensor (49, 6, 66). A continuación se muestra cómo se estructura la información en este formato y cómo se procesa.

```
1 Kismet (1944 film)
2 Kismet is a 1944 Metro-Goldwyn-Mayer film in Technicolor starring Ronald Colman,
     Marlene Dietrich, Joy Page, and Florence Bates.
3 James Craig played the young Caliph of Baghdad, and Edward Arnold was the
     treacherous Grand Vizier.
4 It was directed by William Dieterle, but was not a success at the box office.
5 The film is based on the play of the same name by Edward Knoblock, which was also
     the basis for a 1953 musical.
6 The play had been filmed three times before, in 1914, 1920, and again in 1930 by
     Warner Brothers in an English version directed by John Francis Dillon and in
     a German-language version directed by William Dieterle.

1 Flags of Our Fathers (film)
2 Flags of Our Fathers is a 2006 American war film directed, co-produced and scored
    by Clint Eastwood and written by William Broyles, Jr. and Paul Haggis.
                              . . .
```

Durante el análisis de estas frases, para almacenarlas palabra por palabra en vectores, primero se sustituyen las entidades que se tienen guardadas en el fichero por una clave, después se separan las frases por palabras y por último se vuelve a sustituir la clave por la correspondiente entidad. Esto se hace para evitar que nombres compuestos como pueden ser los títulos de las películas o los directores y actores se consideren como palabras sueltas, es decir se quiere considerar 'Clint Eastwood' como una sola palabra y no considerar 'Clint' y 'Eastwood' como dos palabras independientes.

```
[ [ ['Kismet', '(', '1944', 'film', ')'],
    ['Kismet', 'is', 'a', '1944', 'Metro', '-', 'Goldwyn', '-', 'Mayer', 'film',
         'in', 'Technicolor', 'starring', 'Ronald Colman', ',', 'Marlene Dietrich',
         ',', 'Joy Page', ',', 'and', 'Florence Bates', '.'],
    ['James Craig', 'played', 'the', 'young', 'Caliph', 'of', 'Baghdad', ',',
         'and', 'Edward Arnold', 'was', 'the', 'treacherous', 'Grand', 'Vizier',
```

```
        '.'],
    ['It', 'was', 'directed', 'by', 'William Dieterle', ',', 'but', 'was', 'not',
        'a', 'success', 'at', 'the', 'box', 'office', '.'],
    ['The', 'film', 'is', 'based', 'on', 'the', 'play', 'of', 'the', 'same',
        'name', 'by', 'Edward Knoblock', ',', 'which', 'was', 'also', 'the',
        'basis', 'for', 'a', '1953', 'musical', '.'],
    ['The', 'play', 'had', 'been', 'filmed', 'three', 'times', 'before', ',',
        'in', '1914', ',', '1920', ',', 'and', 'again', 'in', '1930', 'by',
        'Warner', 'Brothers', 'in', 'an', 'English', 'version', 'directed',
        'by', 'John Francis Dillon', 'and', 'in', 'a', 'German', '-', 'language',
        'version', 'directed', 'by', 'William Dieterle', '.']                    ],

  [ ['Flags of Our Fathers', '(', 'film', ')'],
    ['Flags of Our Fathers', 'is', 'a', '2006', 'American', 'war', 'film',
        'directed', ',', 'co', '-', 'produced', 'and', 'scored', 'by',
        'Clint Eastwood', 'and', 'written', 'by', 'William', 'Broyles', ',',
        'Jr', '.', 'and', 'Paul Haggis', '.'],
                                    .  .  .
                                                                              ],
                                          .  .  .
                                                                                 ]
```

De igual manera que con las tareas, se crea un diccionario a partir de todas las palabras presentes en los conjuntos de datos y a cada una se le asigna un índice. Después de esto, cada palabra se sustituye por su índice y se ajustan las dimensiones del tensor.

```
[ [ [121 259 215 167 207 308 309 168   7 222  11   7  12   7 151 147 222  13
     180 133  36 222 150  58 319 194 180  88  65  50 151 222 138  69   9 229
     319 194 180 136  10   0   0   0   0   0   0   0   0   0   0   0   0   0
       0   0   0   0   0   0   0   0   0   0   0   0]
  [121 206 226 160 251 302 259 249 302 276 245 180  57   7 328 323 148 302
   161 211 138  18 244  10   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0]
  [ 77 323 194 180 136   7 179 323 246 138 296 158 302 176 250  10   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0]
  [ 81 260 302 344  39 249  30   7 151  56 323 302 313  71 131  10   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0]
  [ 91 226 138  16 102   9  70   9 100 206 222 120 292 116   7  98   7  89
     7 151  63  10   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0]
  [ 91   5  16 206   6   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
     0   0   0   0   0   0   0   0   0   0   0   0]                       ]
                              .  .  .
                                                                          ]
```

El código principal mostrado en A.1 carga la información en el formato especificado, en este caso en forma de frases. Para ello se hace uso de las funciones en A.3 para analizar el fichero y guardar las palabras en vectores, mediante la función `parse_movies`. La conversión a datos numéricos y vectorización son realizados por la función `vectorize_todo`, conjuntamente con las preguntas como se explicará al final del capítulo, aunque aquí solo se muestra la parte de las frases.

Formato ventana

La información se separa en ventanas de W palabras, donde se considera que la palabra central de la ventana tiene que ser una entidad. En el caso del *KV-MemNN*, se codifica la clave como la ventana entera, y el valor sólo como la palabra central (de igual manera que en las tareas se guardaba la pregunta y su respuesta asociada). Esto tiene sentido ya que es más fácil que la ventana entera se relacione con la pregunta, y la entidad del centro con la respuesta. Por ejemplo, especificando que la ventana sea de 7 palabras, el conjunto de datos se representaría de la siguiente manera:

```
1 1944 film ) Kismet is a 1944        Kismet
2 Kismet in Technicolor starring Ronald Colman , Marlene Dietrich ,   Ronald Colman
3 in Technicolor starring Ronald Colman , Marlene Dietrich ,    Kismet
4 Kismet starring Ronald Colman , Marlene Dietrich , Joy Page ,   Marlene Dietrich
5 starring Ronald Colman , Marlene Dietrich , Joy Page ,      Kismet
                                    .  .  .
```

Después, igual que se hacía con la información en formato frase, se sustituyen las entidades por una clave para así evitar que palabras compuestas se consideren como palabras separadas, y se almacenan en una lista dos elementos: uno es un vector que guarda las palabras de cada ventana y el otro es la palabra que ocupa el centro de la ventana. Por último se revierte el cambio de las palabras por su clave en el diccionario. Así, la información en ventanas queda guardada como sigue:

```
[ [ ( ['1944', 'film', ')', 'Kismet', 'is', 'a', '1944'],    'Kismet' ),
  [ ( ['Kismet', 'in', 'Technicolor', 'starring', 'Ronald Colman', ',',
         'Marlene Dietrich', ','],    'Ronald Colman' ),
  [ ( ['in', 'Technicolor', 'starring', 'Ronald Colman', ',', 'Marlene
        Dietrich', ','],    'Kismet' ),
  [ ( ['Kismet', 'starring', 'Ronald Colman', ',', 'Marlene Dietrich', ',',
         'Joy Page', ','],    'Marlene Dietrich'),
  [ ( ['starring', 'Ronald Colman', ',', 'Marlene Dietrich', ',', 'Joy Page',
         ','], 'Kismet' ),
                                    .  .  .
                                                                          ]
                                  .  .  .
                                                                            ]
```

Una vez se tiene la información disponible como se acaba de mostrar, se sustituirán las palabras por su índice correspondiente en el diccionario para posteriormente almacenar el vector de las palabras de la ventana considerada en la memoria clave y el centro de la ventana en la memoria valor, como se muestra a continuación.

Memoria clave:

```
[ [ [323 130   8 109   8  98   8   0   .  .  .   0]    starring Ronald Colman, ...
    [101 323 130   8 109   8  98   8   .  .  .   0]    Kismet starring  ...
    [247 136 323 130   8 109   8   0   .  .  .   0]    in Techinicolor starring ...
    [101 247 136 323 130   8 109   8   .  .  .   0]    Kismet in Technicolor ...
```

```
    [ 17 228   7 101 251 155  17   0   · · ·   0]    1944 film ) Kismet is a ...
    [  0   0   0   0   0   0   0   0   · · ·   0]
                      .  .  .                  ]
                                                 ]
```

Memoria valor:

```
[ [ [101]      Kismet
    [109]      Marlene Dietrich
    [101]      Kismet
    [130]      Ronald Colman
    [101]      Kismet
    [  0]
      .
      .
      .   ]
   .  .  .
            ]
```

La dimensión del tensor de ventanas para el entrenamiento será inicialmente (5, 23, 8) para la memoria clave, que almacena las ventanas y (5, 23, 1), que almacena los centros de las ventanas. Igual que antes, 5 equivale al número de películas que se están considerando, 23 al número máximo de frases que conforman la información sobre películas una vez generado el fichero con el formato ventana, 8 corresponde al tamaño de la ventana y 1 al tamaño del centro (solo está formado por una palabra). Ya que las dimensiones x y z deben coincidir con las x e y de las preguntas, se redimensionan ambas memorias a (49, 23, 66).

En este caso las funciones utilizadas se encuentran en la sección A.5. El texto completo, mediante la función `parse_windows` se separa en palabras y a partir de aquellas que aparecen en la lista de entidades, se selecciona una ventana de tamaño 7, y se guarda junto con su centro. La conversión a datos numéricos y vectorización son realizados por la función `vectorize_window_todo`, también conjuntamente con las preguntas.

Formato tripleta

El formato tripleta es el propio del texto dispuesto en forma de base de conocimiento. Las tripletas son fragmentos de conocimiento que tienen una estructura de *sujeto-relación-objeto*. El sujeto indica la entidad (personas, organizaciones, lugares, películas, etc.) sobre la que se describe un hecho o relación, el predicado indica el tipo de hecho a describir y el objeto indica un valor adicional que ayuda a completar el hecho. Básicamente, representan un sujeto que realiza una acción sobre un objeto, o si se considera la relación inversa, un objeto que recibe una acción de un sujeto. En el caso del modelo *KV-MemNN* el sujeto y la acción se guardan conjuntamente como la clave, y el objeto asociado a ellos como el valor.

```
1 Kismet directed_by William Dieterle
2 Kismet written_by Edward Knoblock
3 Kismet starred_actors Marlene Dietrich, Edward Arnold, Ronald Colman, James Craig
4 Kismet release_year 1944
5 Kismet in_language English
6 Kismet has_tags bd-r
7 Kismet has_plot Hafiz, a rascally beggar on the periphery of the court of Baghdad,
    schemes to marry his daughter to royalty and to win the heart of the queen of
```

```
     the castle himself.
                                  .  .  .

[ [ ( ['Kismet', 'directed_by'], 'William Dieterle' ),
    ( ['Kismet', 'written_by'], 'Edward Knoblock' ),
    ( ['Kismet', 'starred_actors'], 'Marlene Dietrich, Edward Arnold, Ronald
          Colman, James Craig'),
    ( ['Kismet', 'release_year'], '1944' ),
    ( ['Kismet', 'in_language'], 'English' ),
    ( ['Kismet', 'has_tags'], 'bd-r' ),
    ( ['Kismet', 'has_plot'], 'Hafiz, a rascally beggar on the periphery of the
          court of Baghdad, schemes to marry his daughter to royalty and to win
          the heart of the queen of the castle himself' )                    ]
                           .  .  .
                                                                             ]
```

La información se estructura como en el caso de las ventanas, en una lista que contiene un vector con el sujeto y la acción que realiza, y también el objecto correspondiente.

Memoria clave:

```
[ [ [ 95 226]          Kismet has_plot
    [ 95 227]          Kismet has_tags
    [ 95 234]          Kismet in_language
    [ 95 284]          Kismet release_year
    [ 95 305]          Kismet starred_actors
    [ 95 355]          Kismet written_by
    [ 95 201] ]        Kismet directed_by
    .  .  .
               ]
```

Memoria valor:

```
[ [ [ 74]           Hafiz, a rascally beggar on the periphery of the court ...
    [169]           bd-r
    [ 59]           English
    [ 16]           1944
    [103]           Marlene Dietrich, Edward Arnold, Ronald Colman, James Craig
    [ 58]           Edward Knoblock
    [142] ]         William Dieterle
  .  .  .
           ]
```

La dimensión del tensor de tripletas para el entrenamiento será inicialmente (5, 8, 2) para la memoria clave, que almacena los sujetos y acciones de las tripletas, y (5, 8, 1) para la memoria valor, que almacena los objetos. Como antes, 5 equivale al número de películas que se están considerando, 8 al número máximo de frases que conforman la información sobre películas una vez generado el fichero con el formato tripleta, 2 corresponde al tamaño del vector *[sujeto, acción]* y 1 al tamaño del vector *[objeto]*. Ya que las dimensiones x y z deben coincidir con las x e y de las preguntas, se redimensionan ambas memorias a (49, 8, 66).

La funciones que analizan el texto en forma de tripletas y lo estructuran en la parte clave y valor se muestran en A.4. La conversión a datos numéricos y vectorización son realizados por la función `vectorize_triple_todo`, conjuntamente con las preguntas como en los otros casos.

Preguntas

Las preguntas se procesan de la misma forma que se ha venido explicando, leyéndolas palabra por palabra y sustituyéndolas por su índice correspondiente en el diccionario. Se almacenarán en un tensor de orden dos con dimensiones (49, 66), siendo 49 el número de preguntas y 66 la longitud máxima de frase; y las respuestas en uno de dimensiones (49, 346), siendo 346 el número de palabras en el diccionario (para el caso del formato frase).

```
1 what movies did Edward Knoblock write?    Kismet, The Sin of Madelon Claudet
1 which words describe movie Kismet?    bd-r
1 what year was Kismet released?    1944
1 which person wrote Kismet?    Edward Knoblock
1 what was the popularity rating of Kismet?    unknown
1 what is the language spoken in the film Kismet?    English
1 what is a film directed by William Dieterle?    Kismet
1 who acted in the movie Kismet?    Marlene Dietrich, Edward Arnold, Ronald Colman,
      James Craig
                                  .  .  .
```

```
[ ( ['what', 'movies', 'did', 'Edward Knoblock', 'write'], 'Kismet,
          The Sin of Madelon Claudet' ),
  ( ['which', 'words', 'describe', 'movie', 'Kismet'], 'bd-r' ),
  ( ['what', 'year', 'was', 'Kismet', 'released'], '1944' ),
  ( ['which', 'person', 'wrote', 'Kismet'], 'Edward Knoblock' ),
  ( ['what', 'was', 'the', 'popularity', 'rating', 'of', 'Kismet'], 'unknown' ),
  ( ['what', 'is', 'the', 'language', 'spoken', 'in', 'the', 'film', 'Kismet'],
          'English'),
  ( ['what', 'is', 'a', 'film', 'directed', 'by', 'William Dieterle'], ' Kismet' ),
  ( ['who', 'acted', 'in', 'the', 'movie', 'Kismet'], 'Marlene Dietrich,
          Edward Arnold, Ronald Colman, James Craig' ),
                              .  .  .
                                                                              ]
```

```
[ [326 243 192  57 337   0   · · ·   0]       what movies did Edward Knoblock write
  [328 333 190 242  91   0   · · ·   0]       which words describe movie Kismet
  [326 342 323  91 272   0   · · ·   0]       what year was Kismet released
              .  .  .
                                    ]
```

Por ejemplo, la respuesta a la primera pregunta es 'Kismet, The Sin of Madelon Claudet'. Dado que esa palabra tiene asociado el índice 90 en el diccionario, su representación sería la siguiente:

$$[\underset{1}{0}\ \underset{2}{0}\ \underset{3}{0}\ \cdots\ 0\ 0\ \underset{90}{1}\ 0\ 0\ \cdots\ 0\ 0\ 0]$$

Las preguntas y respuestas se cargan al principio del programa, ya que son comunes a todos los formatos. Son procesadas por la función `parse_questions` de la sección A.2 y vectorizadas en cada caso junto con la información en el formato permitente, como ya se ha dicho, mediante las funciones `vectorize_todo`, `vectorize_window_todo` y `vectorize_triple_todo`.

Codificación interna

A la hora de vectorizar la información, se ha desarrollado una implementación que asocia a cada pregunta las historias (información sobre una película) que contienen alguna información sobre ella y que por tanto permitirán responderla (son las funciones recién mencionadas y disponibles en las secciones A.3, A.5 y A.4). Para esto, lo que se hace es analizar una a una las preguntas y sus respuestas y ver qué entidades aparecen en ellas, guardando las entidades obtenidas en una lista. Posteriormente, se van analizando todas las historias y se guardan en un nuevo tensor aquellas en las que aparecen algunas de las entidades obtenidas para cada una de las preguntas, de forma que a la hora del entrenamiento las correspondientes historias ocupan la misma posición en sus tensores que la que ocupan las preguntas y respuestas. Esto se ha realizado para tener una ordenación de los datos y no tener que analizar repetidamente toda la información completa para cada pregunta, debido a que requiere mucho más tiempo y proporciona peores resultados.

Por ejemplo, habiendo analizado ya tanto las preguntas como la información sobre las películas, dada la primera pregunta y su respuesta

```
Pregunta: ['what', 'movies', 'did', 'Edward Knoblock', 'write']
Respuesta: 'Kismet, The Sin of Madelon Claudet'
```

se crea la lista de entidades presentes tanto en la pregunta como en la respuesta

```
Entidades: ['Edward Knoblock', 'Kismet', 'The Sin of Madelon Claudet']
```

Ahora comienzan a analizarse todas las historias disponibles en el conjunto de datos y se guardan aquellas en cuya primera línea aparece alguna de las entidades que han sido guardadas, ya que esta línea contiene el nombre de la película sobre la que se está aportando información. En este caso, se guardarían dos historias, la que informa sobre la película 'Kismet' y la que informa sobre la película 'The Sin of Madelon Claudet'.

```
[ ['Kismet', '(', '1944', 'film', ')'],
  ['Kismet', 'is', 'a', '1944', 'Metro', '-', 'Goldwyn', '-', 'Mayer', 'film',
     'in', 'Technicolor', 'starring', 'Ronald Colman', ',', 'Marlene Dietrich',
                                    .   .   .                                  ]
                                .   .   .
                                                                              ]

[ ['The Sin of Madelon Claudet'],
  ['The Sin of Madelon Claudet', 'is', 'a', '1931', 'American', 'drama', 'film',
     'directed', 'by', 'Edgar Selwyn', 'and', 'starring', 'Helen Hayes', '.'],
                                .   .   .
                                                                              ]
```

Por último se reemplaza cada palabra por su índice en el diccionario y se almacena en el correspondiente vector en el caso de las preguntas y en las correspondientes memorias en el caso de las historias. Así, el tensor que almacena los elementos con información sobre las películas que será utilizado para el entrenamiento tendrá en cada posición la información guardada que puede responder a las preguntas almacenadas en cada posición del tensor que alberga la información sobre las preguntas.

Capítulo 3

Redes neuronales y de memoria

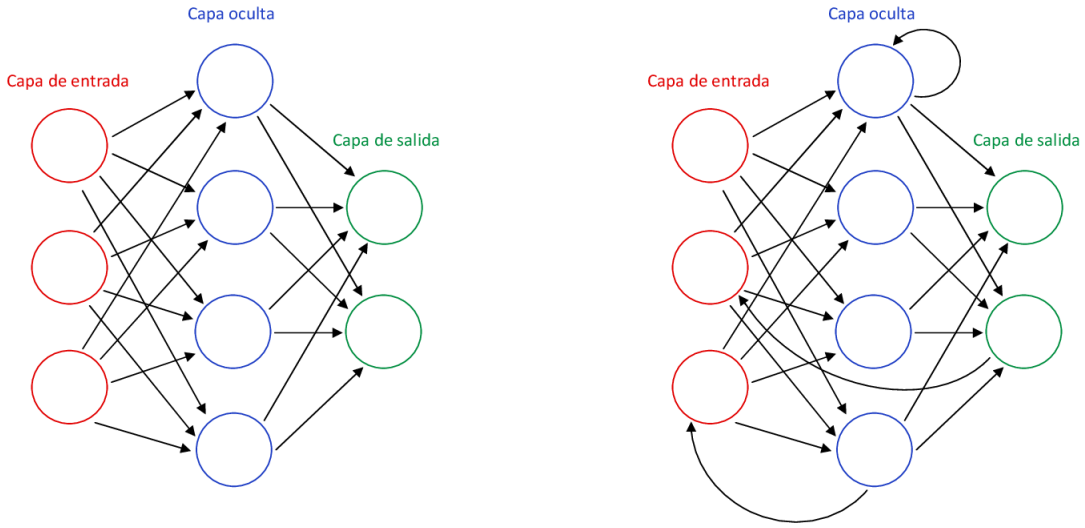
« ... lo que queremos es una máquina que pueda aprender de la experiencia. »
— Alan Turing

En este capítulo se exponen primero los conceptos básicos sobre redes neuronales artificiales, y en particular sobre redes neuronales recurrentes, que son con las que se ha trabajado, ya que son especialmente útiles para el procesamiento secuencial de datos como sonido, series de tiempo, o lenguaje natural escrito, como es el caso de este trabajo. Asimismo, se profundiza en los algoritmos que permiten mejorar su funcionamiento y también en casos particulares que son ampliación de estos modelos. Adicionalmente se introducen las redes de memoria, que han sido el propósito de estudio en este trabajo. Su interés recae sobre todo en la extensión de estas que surge de su combinación con las redes recurrentes, cuando la recurrencia aparece en alguna de sus componentes. Se presentan los parámetros principales para el entrenamiento de las redes, y los dos modelos particulares de redes neuronales de memoria estudiados.

3.1. Redes Neuronales Artificiales

Las redes neuronales artificiales (*artificial neural networks, ANN*) son algoritmos matemáticos surgidos de la idea de imitar el comportamiento de las redes neuronales biológicas, en particular las humanas. Siguiendo esta simulación del cerebro humano, las redes estarán formadas por un conjunto de neuronas artificiales (nodos). Están distribuidas en diferentes capas, interconectadas entre sí, siguiendo una configuración determinada [16]. Una red contará al menos con dos capas, una de entrada y otra de salida, y además podrán existir también capas ocultas, que influirán en la complejidad de su configuración. Cada conexión entre dos neuronas tendrá un peso asociado, indicador de la importancia de dicha conexión, y que se utilizará para ponderar los datos de entrada (*inputs*) de la correspondiente neurona junto con un sesgo. Estos pesos deberán estimarse a partir del conjunto de datos; es lo que se denomina proceso de aprendizaje o entrenamiento de la red neuronal.

Las redes neuronales, respecto a su configuración, como puede verse en la figura 3.1, se diferencian en redes de propagación hacia adelante o *feedforward* (no existe retroalimentación) y redes recurrentes, dependiendo de si la salida de una neurona solo puede ser la entrada de neuronas de la capa siguiente o por el contrario la salida de una neurona puede ser también entrada de neuronas de capas interiores o de la misma capa (ella misma incluida).



(a) Red neuronal feedforward (b) Red neuronal recurrente

Figura 3.1: Tipos de redes neuronales

La motivación de estas redes es poder contar con una 'memoria' en la que pueda almacenarse información previa, así como acceder múltiples veces y razonar sobre ella para imitar más fielmente lo que sería una red neuronal biológica. Esta recurrencia puede interpretarse como la existencia de múltiples capas en la red donde la información se transmite de capa a capa, como se ve en la figura 3.2.

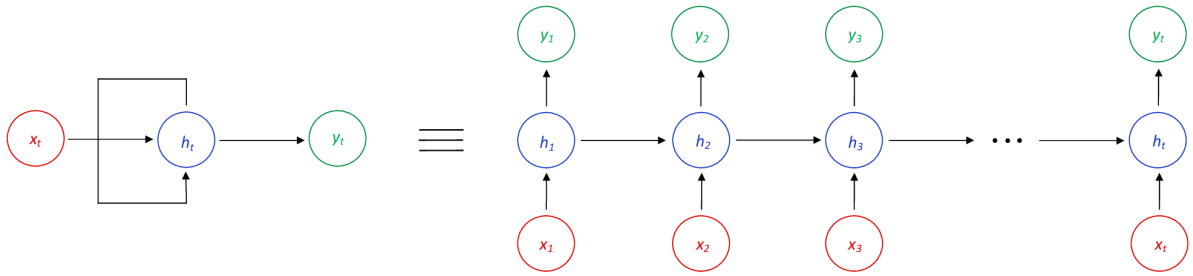


Figura 3.2: Esquema de recurrencia en una red reuronal

Suponer, sin pérdida de generalidad, que se tiene una red neuronal recurrente con solo tres capas: entrada, oculta y salida, y cuya capa oculta está conectada consigo misma además de con la capa de salida. Sea n, m y l el número de neuronas existentes en la capa de entrada, oculta y salida, respectivamente. Sea t el indicador de los tiempos de observación. Se definen

- x_t^i las entradas (*inputs*) de la red en el instante t, $i = 1, \ldots, n$
- h_t^j las salidas (*outputs*) de la capa oculta en el instante t, $j = 1, \ldots, m$
- y_t^k las salidas finales de la red en el instante t, $k = 1, \ldots, l$
- f y g las funciones de salida de la capa oculta y de la capa de salida, respectivamente
- w_{ij}^1 los pesos asociados a las conexiones entre las neuronas de la capa de entrada y la oculta
- w_{jh}^2 los pesos asociados a las conexiones entre las las propias neuronas de la capa oculta
- w_{jk}^3 los pesos asociados a las conexiones entre las neuronas de la capa oculta y la de salida

- θ_j^1 y θ_k^2 los sesgos (*bias*) asociados a la neurona j de la capa oculta y la neurona k de la capa de salida

Puede definirse la red neuronal recurrente mediante la siguiente expresión:

$$h_t^j = f\left(\sum_{i=1}^{n} w_{ij}^1 x_t^i + \sum_{h=1}^{m} w_{jh}^2 h_{t-1}^h + \theta_j^1\right), \quad y_t^k = g\left(\sum_{j=1}^{l} w_{jk}^3 h_t^j + \theta_j^2\right).$$

De ahora en adelante se utilizará la siguiente notación matricial para simplificar las fórmulas que definen las redes. Esta representación es como considerar un solo nodo por cada capa.

- $x_t = (x_t^1, \ldots, x_t^n)$
- $h_t = (h_t^1, \ldots, h_t^m)$
- $y_t = (y_t^1, \ldots, y_t^l)$
- $W^1 = (w_{ij}^1)$ la matriz de los pesos asociados a las conexiones entre las neuronas de la capa de entrada y la oculta
- $W^2 = (w_{jh}^2)$ la matriz los pesos asociados a las conexiones entre las propias neuronas de la capa oculta
- $W^3 = (w_{jk}^3)$ los pesos asociados a las conexiones entre las neuronas de la capa oculta y la de salida
- $\theta^1 = (\theta_j^1)$ y $\theta^2 = (\theta_k^2)$ las matrices (vectores) de los sesgos (*bias*) asociados a la neurona j de la capa oculta y la neurona k de la capa de salida

Así, con esta notación, puede definirse la red neuronal recurrente mediante la siguiente expresión:

$$h_t = f(W^1 x_t + W^2 h_{t-1} + \theta^1), \quad y_t = g(W^3 h_t + \theta^2).$$

Si existen varias capas ocultas, una capa recibe como entrada los *inputs* iniciales además de la salida de la capa anterior, y la capa de salida final recibe como entrada las salidas de las capas ocultas [17]. Esto hace más fácil el entrenamiento de redes profundas ya que reduce el número de pasos de procesamiento de abajo a arriba de la red, y permite así suavizar el problema del 'desvanecimiento del gradiente'. Una representación de una red con varias capas ocultas como acaba de explicarse puede verse en la figura 3.3.

3.1.1. Retropropagación y Gradiente Descendente

El algoritmo de retropropagación (*backpropagation*) [3] es un algoritmo de aprendizaje que permite adaptar y modificar los parámetros de la red neuronal durante el entrenamiento. Esta adaptación de dichos parámetros (pesos) se consigue minimizando una función de pérdida, que mide el error que se comete con la salida generada respecto de la salida real (etiquetas).

El nombre se debe a que el error es propagado a través de la red neuronal desde la capa de salida hasta la capa de entrada. Esto permite que los pesos sobre las conexiones de las neuronas ubicadas en las capas ocultas cambien durante el entrenamiento. El cambio de los pesos en las conexiones de las neuronas, además de influir sobre la entrada global, influye en la activación y por consiguiente en la salida de una neurona.

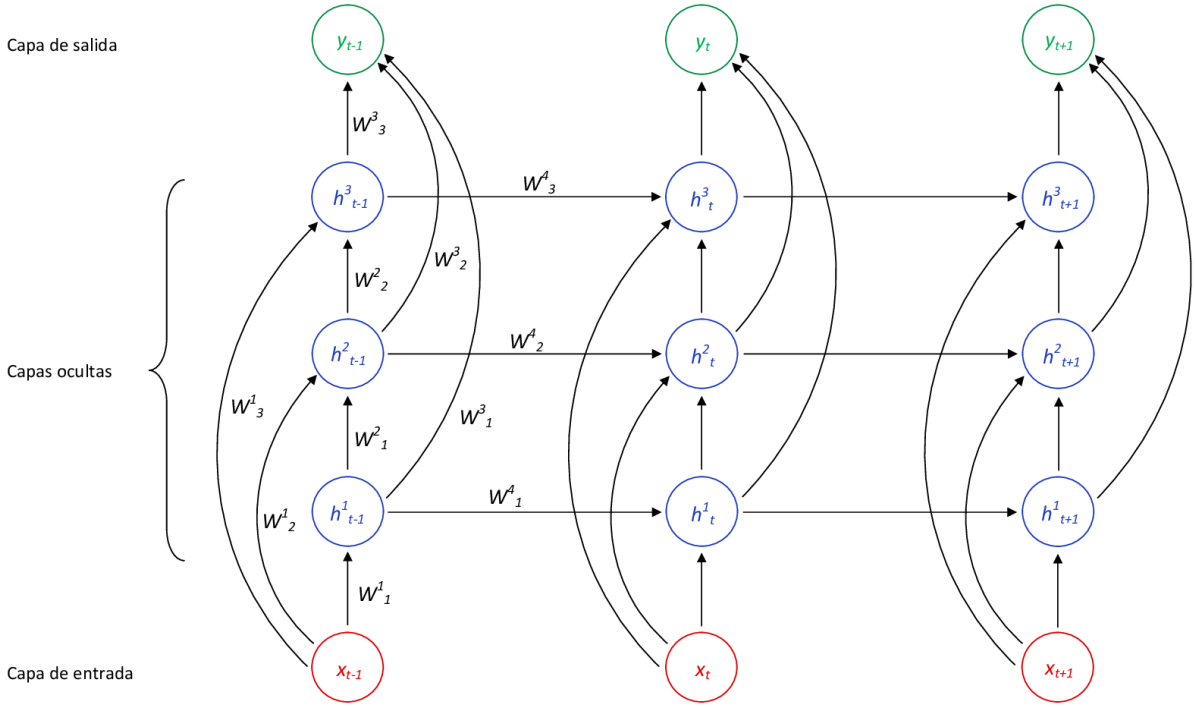


Figura 3.3: Esquema de red neuronal recurrente profunda (3 capas ocultas)

El método que utiliza el algoritmo de retropropagación original para la minimización del error es el método iterativo de primer orden conocido como gradiente descendente *(Gradient Descent, GD)*, que consigue una adaptación de los pesos siguiendo la dirección de búsqueda negativa del gradiente de la función del error en el espacio de pesos, de ahí el nombre, y que determina la zona donde la disminución del error es más rápida.

Para el desarrollo del método se considera una red neuronal simplificada como la mostrada en la figura 3.4.

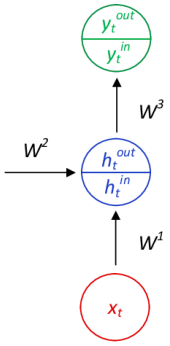


Figura 3.4: Estructura de red neuronal simplificada.

Para las neuronas de la capa oculta y de la capa de salida se definen variables de entrada y salida en cada instante de tiempo t: h_t^{in}, h_t^{out}, y_t^{in}, y_t^{out}.

Puede suponerse, sin pérdida de generalidad, el error cuadrático como función del error a minimizar:

$$e_{total} = \frac{1}{2}(a_t - y_t^{out})^2, \tag{3.1}$$

siendo y_t^{out} la salida predicha en el nodo y_t y a_t la salida real.

A continuación se calcula cuánto afecta al error total un cambio en cualquiera de los pesos, es decir $\frac{\partial e_{total}}{\partial W^i}$ (el gradiente con respecto a W^i, para $i = 1,2,3$).

$$\frac{\partial e_{total}}{\partial W^3} = \frac{\partial e_{total}}{\partial y_t^{out}} \frac{\partial y_t^{out}}{\partial y_t^{in}} \frac{\partial y_t^{in}}{\partial W^3}$$

$$\frac{\partial e_{total}}{\partial y_t^{out}} = -(a_t - y_t^{out})$$

$y_t^{out} = g(y_t^{in}) \Rightarrow \frac{\partial y_t^{out}}{\partial y_t^{in}} = g'(y_t^{in})$, depende de cuál sea la función de activación g

$y_t^{in} = W^3 h_t^{out} + \theta^2 \Rightarrow \frac{\partial y_t^{in}}{\partial W^3} = h_t^{out}$

luego

$$\frac{\partial e_{total}}{\partial W^3} = -(a_t - y_t^{out})\ g'(y_t^{in})\ h_t^{out}. \tag{3.2}$$

$$\frac{\partial e_{total}}{\partial W^1} = \frac{\partial e_{total}}{\partial h_t^{out}} \frac{\partial h_t^{out}}{\partial h_t^{in}} \frac{\partial h_t^{in}}{\partial W^1}$$

$$\frac{\partial e_{total}}{\partial h_t^{out}} = \frac{\partial e_{total}}{\partial y_t^{in}} \frac{\partial y_t^{in}}{\partial h_t^{out}} = \frac{\partial e_{total}}{\partial y_t^{out}} \frac{\partial y_t^{out}}{\partial y_t^{in}} \frac{\partial y_t^{in}}{\partial h_t^{out}} = -(a_t - y_t^{out})\ g'(y_t^{in})\ W^3$$

$h_t^{out} = f(h_t^{in}) \Rightarrow \frac{\partial h_t^{out}}{\partial h_t^{in}} = f'(h_t^{in})$, depende de cuál sea la función de activación f

$h_t^{in} = W^1 x_t + W^2 h_{t-1}^{out} + \theta^1 \Rightarrow \frac{\partial h_t^{in}}{\partial W^1} = x_t$

luego

$$\frac{\partial e_{total}}{\partial W^1} = -(a_t - y_t^{out})\ g'(y_t^{in})\ W^3\ f'(h_t^{in})\ x_t. \tag{3.3}$$

$$\frac{\partial e_{total}}{\partial W^2} = \frac{\partial e_{total}}{\partial h_t^{out}} \frac{\partial h_t^{out}}{\partial h_t^{in}} \frac{\partial h_t^{in}}{\partial W^2}$$

$$h_t^{in} = W^1 x_t + W^2 h_{t-1}^{out} + \theta^1 \Rightarrow \frac{\partial h_t^{in}}{\partial W^2} = h_{t-1}^{out}$$

luego

$$\frac{\partial e_{total}}{\partial W^2} = \frac{\partial e_{total}}{\partial h_t^{out}} \frac{\partial h_t^{out}}{\partial h_t^{in}} \frac{\partial h_t^{in}}{\partial W^2} = -(a_t - y_t^{out})\ g'(y_t^{in})\ W^3\ f'(h_t^{in})\ h_{t-1}^{out}. \tag{3.4}$$

Por último se lleva a cabo la actualización de los pesos mediante:

$$W^i = W^i - \eta \frac{\partial e_{total}}{\partial W^i}, \qquad \text{para } i = 1,2,3 \tag{3.5}$$

siendo η la tasa de aprendizaje seleccionada.

Esta actualización se realizará de forma iterativa hasta alcanzar el criterio de parada establecido.

En lugar del error cuadrático, se considerará en los modelos la función pérdida de entropía cruzada:

$$e_{total} = -a_t \; log \; y_t + (1 - a_t) \; log(1 - y_t) \tag{3.6}$$

En algunos de los modelos que se introducirán a continuación se utiliza el método del gradiente descendente estocástico (*Stochastic Gradient Descent, SGD*), también conocido como gradiente descendente incremental, que es una aproximación estocástica del método original. Para este método es necesario solo una muestra del conjunto de entrenamiento para actualizar un parámetro, mientras que en el gradiente descendente se necesita el conjunto de entrenamiento completo para realizar la actualización, lo que puede llevar mucho tiempo si dicho conjunto de entrenamiento es muy grande. La función del error no se minimizará tanto como en el caso del *GD*, pero el *SGD* convergerá más rápido.

Adam: Adaptative Moment Estimation

Es un algoritmo de optimización que utiliza, además de los gradientes, sus momentos de segundo orden [21]. La actualización de los pesos es en este caso

$$W^i = W^i - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}, \tag{3.7}$$

siendo η la tasa de aprendizaje y ε un número pequeño para prevenir la división por cero

$\hat{m}_t = \dfrac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \dfrac{v_t}{1 - \beta_2^t}$ las correcciones sesgadas de los momentos de primer y segundo orden

$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \dfrac{\partial e_{total}}{\partial W^i}, \quad v_t = \beta_2 m_{t-1} + (1 - \beta_2) \left(\dfrac{\partial e_{total}}{\partial W^i} \right)^2$ las estimaciones sesgadas de los momentos de primer y segundo orden

y β_1, β_2 las tasas de descrecimiento de las estimaciones de los momentos.

Problema del desvanecimiento del gradiente

Ciertas funciones de activación, como pueden ser la función sigmoide y la tangente hiperbólica,

$$y_t^{out} = \sigma(y_t^{in}) = \frac{1}{1 + e^{-y_t^{in}}}, \qquad y_t^{out} = \varsigma(y_t^{in}) = \frac{e^{y_t^{in}} - e^{-y_t^{in}}}{e^{y_t^{in}} + e^{-y_t^{in}}} \tag{3.8}$$

generan problemas en redes neuronales que utilizan métodos basados en el gradiente, como por ejemplo la retropropagación. Estos problemas dificultan el aprendizaje y la elección de los parámetros adecuados en las primeras capas de la red, ya que estos se vuelven muy pequeños (la función sigmoide transforma la recta real a un rango [0, 1] y la tangente hiperbólica a [-1, 1]), causando que la red caiga en un mínimo local. Además esto empeora en función del número de capas ocultas.

3.1.2. Redes de Memoria a Largo y Corto Plazo (LSTM)

Como se ha introducido, las redes *LSTM* son redes neuronales recurrentes cuyas neuronas en las capas ocultas son reemplazadas por celdas de memoria [4] [18] [19]. Estas celdas cuentan con un 'sistema de compuertas' que deciden la información que debe almacenarse en la memoria actual, la que debe olvidarse y la que se transmitirá al resto de capas. Las funciones de activación consideradas generalmente en estas redes son la sigmoide y la tangente hiperbólica. La estructura de las capas ocultas cuando se consideran redes *LSTM* puede representarte, para un instante de tiempo t, como se muestra en la figura 3.5, mediante las expresiones siguientes:

$$m_t = f_t * m_{t-1} + g_t * i_t, \qquad \text{memoria interna}$$

$$i_t = \sigma(W_1^i x_t + W_2^i h_{t-1}), \qquad \text{compuerta de entrada}$$

$$f_t = \sigma(W_1^f x_t + W_2^f h_{t-1}), \qquad \text{compuerta de olvido}$$

$$o_t = \sigma(W_1^o x_t + W_2^o h_{t-1}), \qquad \text{compuerta de salida}$$

$$g_t = \varsigma(W_1^g x_t + W_2^g h_{t-1}), \qquad \text{entrada al estado oculto}$$

$$h_t = \varsigma(m_t) * o_t, \qquad \text{salida del estado oculto}$$

donde x_t es el vector de entrada a la capa correspondiente en el instante t, σ representa la función sigmoide, ς la función tangente hiperbólica, y W_1, W_2 son las matrices de pesos a estimar. En cada instante de tiempo, a partir de x_t, la *LSTM* genera los correspondientes h_t y m_t.

Nota: Aunque la *LSTM* original no contaba con compuerta de olvido, esta extensión es la considerada como red *LSTM* básica, ya que permite olvidar los bloques de memoria que ya no se utilizan, sin perder las ventajas de la *LSTM*.

3.1.3. Celdas GRU

Las unidades recurrentes cerradas (*Gated Recurrent Units, GRU*) tienen menos parámetros que las *LSTM*. Combinan las compuertas de entrada y olvido en una única compuerta denominada de actualización, y también la memoria interna con el estado oculto. Su estructura, representada en la figura 3.6 viene dada por las siguientes expresiones:

$$z_t = \sigma(W_1^z x_t + W_2^z h_{t-1}), \qquad \text{compuerta de actualización}$$

$$r_t = \sigma(W_1^r x_t + W_2^r h_{t-1}), \qquad \text{compuerta de restauración}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \varsigma(W_1^h x_t + W_2^h (r_t * h_{t-1})), \qquad \text{salida del estado oculto}$$

donde x_t es el vector de entrada a la capa correspondiente en el instante t, σ representa la función sigmoide, ς la función tangente hiperbólica, y W_1, W_2 son las matrices de pesos a estimar. En cada instante de tiempo, a partir de x_t, la *GRU* genera el correspondientes h_t.

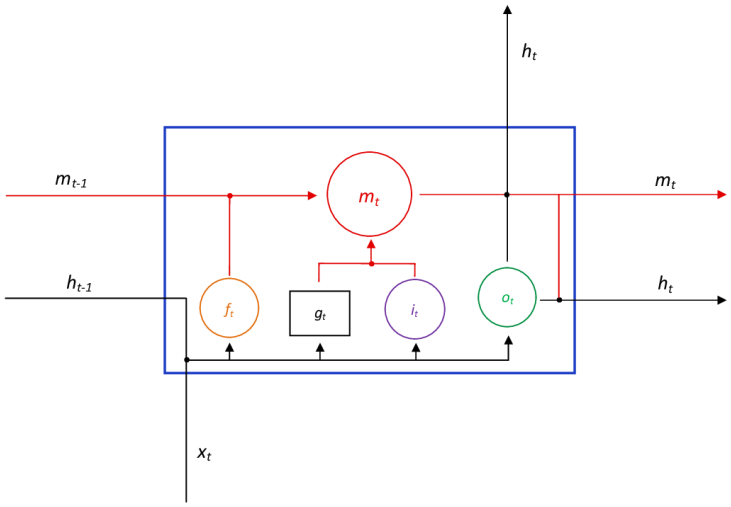


Figura 3.5: Esquema de una celda *LSTM*.

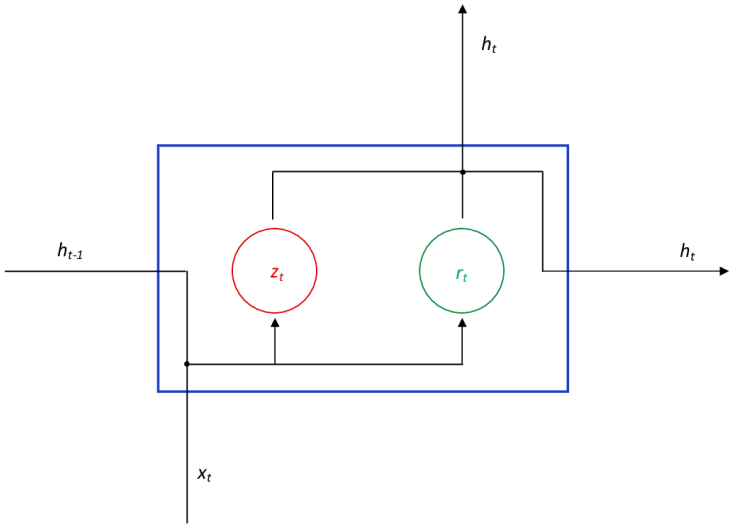


Figura 3.6: Esquema de una celda *GRU*.

3.2. Redes de memoria

Las redes de memoria, como su propio nombre indica, constan de una memoria $\mathbf{m}$ (un array de objetos denotados como $\mathbf{m}_i$) sobre la que se puede leer y escribir, y cuatro componentes. El esquema general de estas redes se muestra en la figura 3.7. La componente de entrada, I, es la encargada de codificar la entrada x a la representación interna pertinente, $I(x)$. La componente de generalización, G, actualiza todas las memorias m_i dada la nueva entrada: $m_i = G(m_i, I(x), \mathbf{m}), \forall i = 1, \ldots, N$. La componente de salida, O, se encarga de procesar una salida o a partir de la nueva entrada $I(x)$ y la $\mathbf{m}$, $o = O(I(x), \mathbf{m})$. Por último, la componente de respuesta R decodifica la salida o dando una respuesta r interpretable como puede ser un texto o una acción: $r = R(o)$. En cada una de estas componentes puede utilizarse cualquier modelo conocido en la literatura de Machine Learning, pero el estudio aquí se centrará en el caso de que alguna de las componentes sea una red neuronal recurrente, obteniéndose así las redes neuronales de memoria.

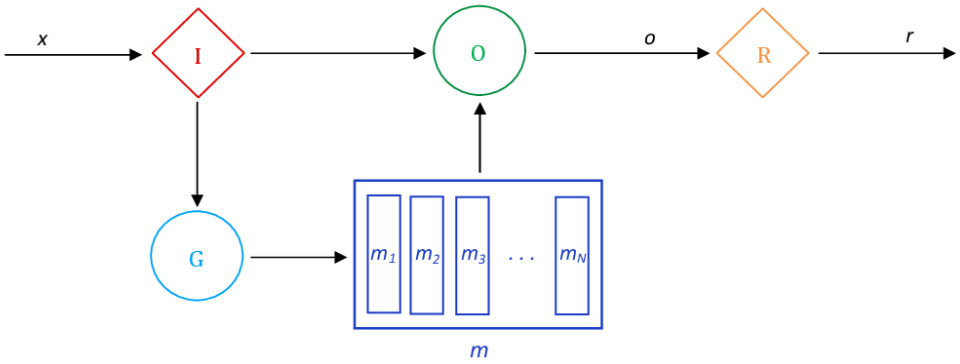


Figura 3.7: Esquema de una red de memoria.

En el modelo básico, I toma una entrada x que puede codificarse a una representación interna y pasa a la componente G que se encargará de almacenarla en la siguiente ranura de memoria que haya disponible. La red neuronal se introduce en la componente O, que en el caso de búsqueda de respuestas, se encarga de encontrar las k frases soporte que responden a la pregunta en cuestión, con $k \leq N$, siendo N el número de celdas en la memoria. R generará una respuesta a partir de la entrada x (pregunta) y las memorias soporte seleccionadas. Para k recurrencias el esquema de la red neuronal de memoria es el que se detalla a continuación y se representa en la figura 3.8.

Cuando ya se han almacenado todas las frases en memoria y la siguiente entrada x es la pregunta cuya respuesta quiere obtenerse (suponer $I(x) = x$), el módulo O (ver como $O_1, \ldots, O_k$) combina esta entrada con todas las memorias m_i y les asigna unas calificaciones (función s_O) según la relevancia de dichas memorias para responder a la pregunta, y después se selecciona la memoria cuyo índice ha sido el mayor. Se genera la salida o_1 formada por la entrada y la memoria que proporcionaba la mayor calificación.

$$j_1 = \arg\max_{i=1,\ldots,N} s_O(x, m_i), \qquad o_1 = O_1(x, \mathbf{m}) = [x, m_{j_1}]. \tag{3.9}$$

Para $k \geq 2$, el proceso es el mismo solo que ahora la función s_O combina el conjunto de la entrada y las memorias ya seleccionadas en las anteriores vueltas, con las memorias que quedan todavía sin seleccionar.

$$j_k = \arg\max_{i=1,\ldots,N} s_O([x, m_{j_1}, \ldots, m_{j_{k-1}}], m_i), \qquad o_k = O_k(x, \mathbf{m}) = [x, m_{j_1}, \ldots, m_{j_k}]. \tag{3.10}$$

Por último, el módulo R genera una respuesta que puede ser simplemente la última memoria seleccionada, pero que posiblemente no sea una respuesta con sentido. Para general una respuesta lógica, lo que se hace es de nuevo combinar la última salida o_k con todas las palabras que aparecían en el texto (W), asignarles una calificación (función s_R) y seleccionar la palabra asociada a la mayor calificación:

$$r = \arg\max_{w \in W} s_R(o_k, w). \tag{3.11}$$

Las funciones s_O y s_R son de la forma de un modelo de inmersión:

$$s(x, y) = \Phi_x(x)^T U^T U \Phi_y(y), \tag{3.12}$$

donde U es una matriz $n \times D$, con D el número de características y n la dimensión de inmersión. Las funciones Φ_x y Φ_y transforman el texto original al espacio de variables D-dimensional. Las matrices de pesos que usa cada función, U_O y U_R, son distintas.

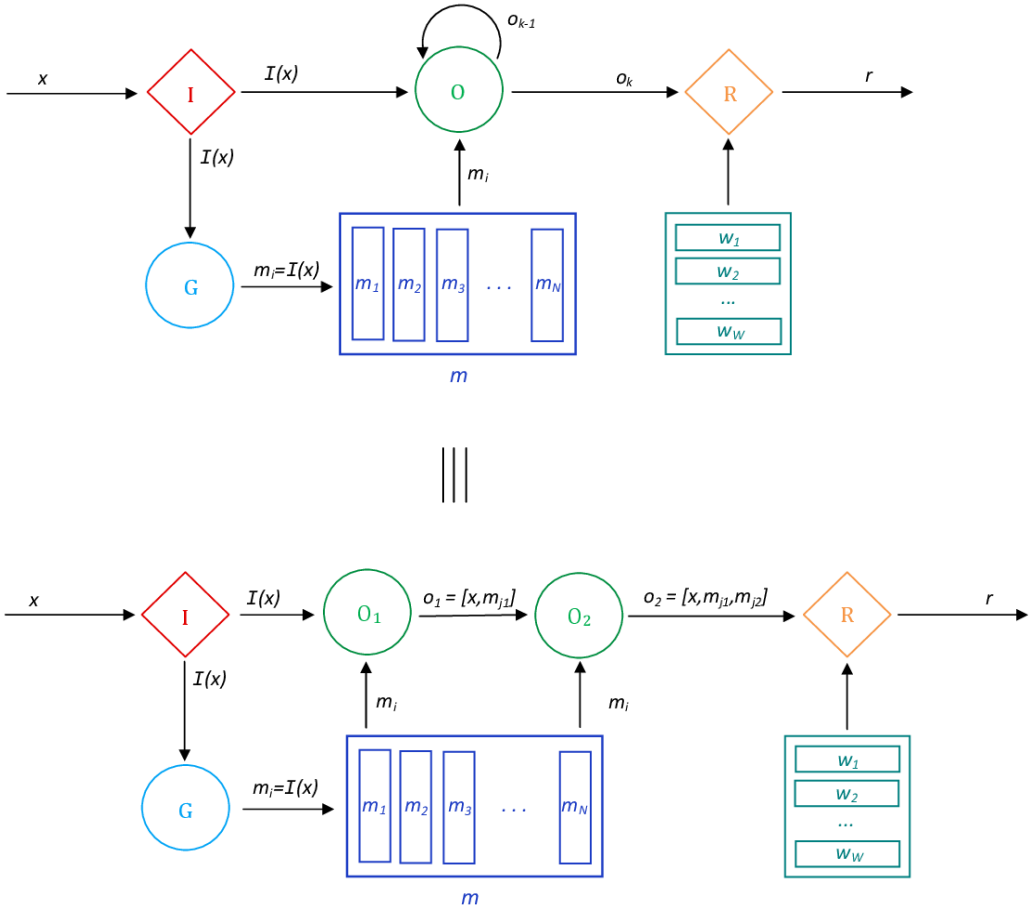


Figura 3.8: Esquema de una red neuronal de memoria para $k = 2$.

El entrenamiento del modelo es supervisado donde para cada pregunta se da la respuesta correspondiente, y además se conocen las frases soporte que permiten responder dicha pregunta. Se lleva a cabo minimizando el error cometido con la predicción realizada mediante el método del gradiente descendiente estocástico. Para el caso $k = 2$, dados una pregunta x, su respuesta r y sus frases soporte almacenadas en m_{j_1} y m_{j_2}, minimizamos sobre los parámetros del modelo U_O y U_R:

$$\sum_{\overline{f} \neq m_{j_1}} \max(0, \gamma - s_O(x, m_{j_1}) + s_O(x, \overline{f})) + \sum_{\overline{f}' \neq m_{j_2}} \max(0, \gamma - s_O([x, m_{j_1}], m_{j_2}) + s_O([x, m_{j_1}], \overline{f}')) +$$

$$+ \sum_{\overline{r} \neq r} \max(0, \gamma - s_O([x, m_{j_1}, m_{j_2}], r) + s_O([x, m_{j_1}, m_{j_2}], \overline{r})) \tag{3.13}$$

donde $\overline{f}$, $\overline{f}'$ y $\overline{r}$ son el resto de opciones que no son las correctas, y γ es el rendimiento.

3.2.1. Parámetros del entrenamiento

Para el entrenamiento de la red neuronal, además de fijar una función del error a minimizar y la tasa de aprendizaje para aplicar el método del gradiente descendente, son necesarios otros parámetros, como unos valores iniciales de los pesos, con los que comenzar a entrenar la red, y diferentes funciones de

activación para las neuronas. También tendrá que establecerse algún criterio de parada del entrenamiento, que será por ejemplo un número determinado de épocas de ejecución. Se entiende por época cada pasada que la red realiza sobre el conjunto de datos completo.

La correcta elección y combinación de estos parámetros influirá notablemente en el entrenamiento de la red y por tanto en la obtención de mejores resultados. Es por ello que se ha dedicado un capítulo al análisis de los resultados en función de los parámetros.

Vector de pesos inicial

Para el entrenamiento de la red es necesario, como se ha adelantado anteriormente, un valor inicial en los pesos. Esta inicialización se lleva a cabo aleatoriamente. Existen diferentes métodos para llevar a cabo esta inicialización previa al entrenamiento y que pueden influir en el número de iteraciones necesarias para que la red converja. Uno de los utilizados en los entrenamientos es el denominado *Xavier* [22]. Con este método se pretende dar valores iniciales aleatorios a los pesos, que no sean ni muy grandes ni muy pequeños, y evitar así saturaciones en la red. Consiste en hacer que la varianza de las entradas de las neuronas sea la unidad. Para asegurar esto, considerando que los datos de entrada tienen media 0 y varianza 1 (pueden normalizarse), y que son independientes, pueden tomarse pesos aleatorios con varianza inversamente proporcional al número de entradas.

Los pesos pueden seguir una distribución uniforme o normal:

$$W^i \sim U\left[-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right], \quad W^i \sim N\left(0, \sqrt{\frac{2}{n_{in}+n_{out}}}\right),$$

siendo n_{in} y n_{out} el número de neuronas de las capas conectadas por los pesos en cuestión.

Algoritmo de optimización

Otra posible elección para la comparación de resultados es el algoritmo de optimización utilizado en la retropropagación. Se compararán los que se han explicado en la sección 3.1.1, el gradiente descendente estocástico y el *Adam*.

Tasa de aprendizaje

La tasa de aprendizaje, como se ha visto en 3.1.1 al explicar el algoritmo de retropropagación, influye en la variación de los pesos: a mayor tasa de aprendizaje más cambio sufren estos pesos. Si se elije una tasa de aprendizaje pequeña las actualizaciones de los pesos son más suaves, pero también el aprendizaje por parte de la red es más lento. Por otro lado, si la tasa de aprendizaje es demasiado grande se pueden producir oscilaciones bruscas del error o caer en un mínimo local.

La tasa de aprendizaje puede ir variando a lo largo del entrenamiento, para así evitar el sobreentrenamiento. Una opción es hacerlo mediante el algoritmo *Adam* como ya se ha explicado. En los modelos que se presentarán a continuación, lo que se hace también es establecer un número de épocas tras el cual se realizará una disminución de la tasa de aprendizaje. También puede hacerse aplicando una función de decrecimiento exponencial.

Función de activación

En las redes que se van a presentar, se define la función $Softmax$ o función exponencial normalizada:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}.$$

Puede interpretarse como una función que asigna una probabilidad ya que proporciona valores entre 0 y 1, y la suma total de los mismos es 1. Es utilizada en la mayoría de los problemas de clasificación. En todos los modelos estudiados esta función se encuentra en las capas ocultas para asignar una probabilidad de coincidencia a las frases que van a almacenarse en memoria con respecto a la pregunta en cuestión, y también en la capa de salida para generar la estimación de la respuesta. Recordar que las respuestas se representaban mediante vector *one-hot*, luego tiene sentido que la estimación de la pregunta se represente mediante la función *Softmax* que como se ha dicho los resultados que genera están entre 0 y 1.

Tipos de neuronas

Para el estudio del modelo de red neuronal recurrente, uno de los parámetros que puede modificarse para la obtención de diferentes resultados es el tipo de neuronas que conforman la red. Se entrenará la red con tres tipos diferentes de neuronas, entendidas como recombinaciones de variables: la celda básica, la celda *LSTM* y la *GRU*, explicadas estas últimas en las secciones 3.1.2 y 3.1.3 respectivamente.

Épocas

Las épocas representan el número de veces que la red hace una pasada sobre todo el conjunto de datos. Se indicará previamente al entrenamiento el número de épocas que quieren considerarse. En las pruebas realizadas se han considerado mínimo 60 épocas y máximo 200.

Batch

Se divide el conjunto de entrenamiento en grupos de menor tamaño, es útil cuando se trabaja con conjuntos de datos muy grandes. Cada uno de estos grupos recibe el nombre de *minibatch* o *batch* por simplificación. La red actualiza los pesos tras pasar por cada uno de los grupos, por lo que se acelera la convergencia en contraposición a si los pesos tuviesen que actualizarse tras pasar por todos los datos de entrenamiento. En este caso una época concluye cuando la red ha pasado por todos los *batches*.

Hops

El número de *hops* o saltos es un parámetro que establece el número de veces que se accede y se lee la memoria. Estos saltos (pueden verse como iteraciones) permiten buscar información relevante almacenada en diferentes ranuras de memoria para dar una mejor respuesta, por ejemplo cuando son necesarias dos o más frases soporte para responder a una pregunta, como se mostraba en la figura 3.8 para dos saltos.

Tamaño de inmersión

A la hora de transformar los datos de entrada por inmersión a un espacio de dimensión d como se explicó en la sección 2.1, esta trasformación se realiza multiplicando sus correspondientes representaciones vectoriales por una matriz, la llamada *matriz de inmersión*. Esta matriz se corresponde con los pesos de la red entre la capa de entrada y la capa oculta, y se irá estimando a lo largo del entrenamiento.

El tamaño de la inmersión d se establece antes de comenzar el entrenamiento y es uno de los parámetros que se ha ido variando para la comparación de resultados del capítulo 4. Establece el número de neuronas que componen la capa oculta.

Tamaño de memoria

La capacidad de almacenaje de la memoria está restringida a las n frases más recientes, este valor es establecido previo al entrenamiento. Dado que para cada problema el número de frases y el número de palabras en cada frase varía, se utiliza un símbolo nulo para rellenarlas y que todas ellas tengan un mismo tamaño fijado. Este tamaño se establecerá durante el procesamiento de la información, ya que será el máximo entre el número de palabras de todas las frases y de todas las preguntas.

Tamaño de la norma del gradiente

Es habitual limitar el tamaño del gradiente cuando se aplica el método de retropropagación en redes neuronales, para evitar que no tome valores demasiado grandes ni pequeños, y conseguir que el método del gradiente descendente estocástico funcione mejor.

3.2.2. Red de Memoria End-to-End (MemN2N)

Esta red sigue la estructura de la red de memoria introducida al comienzo de la sección 3.2, con la diferencia de que está entrenada de extremo a extremo (*end-to-end*). Esta estrategia de entrenamiento consiste en entrenar todo el sistema a la vez, es decir todos los módulos al mismo tiempo desde el de entrada al de salida, de ahí el nombre. Esto hace que la red requiera significativamente menos supervisión durante el entrenamiento, realizado mediante retropropagación, haciéndola más aplicable a escenarios realistas. En este modelo la recurrencia se encuentra en las múltiples lecturas que realiza la red sobre la memoria antes de generar una salida. En la figura 3.9 se ilustra el esquema de esta red a partir del que se mostró para la red de memoria básica. A continuación se detalla su funcionamiento.

Sea $x_1, \ldots, x_n$ el conjunto de datos de entrada que deben almacenarse en la memoria, q la pregunta y a su respuesta. El modelo consta de tres fases:

1. Se codifican los datos de entrada $\{x_i\}$ en vectores de memoria $\{m_i\}$ de dimensión d, mediante una matriz A, y la pregunta q se codifica mediante una matriz B en el vector u. A continuación, a cada memoria m_i se le asigna una probabilidad p_i al comparar la pregunta con la entrada:

$$m_i = Ax_i, \quad u = Bq, \quad p_i = Softmax(u^T m_i).$$

2. Igualmente se codifican los x_i como vectores de salida c_i mediante una matriz C, y se devuelve el vector de la suma ponderada de las salidas c_i mediante sus probabilidades asociadas:

$$c_i = Cx_i, \quad o = \sum_i p_i c_i.$$

3. Por último, se genera la predicción final $\hat{a}$ mediante la suma del vector o y la pregunta codificada u, multiplicada por una matriz de pesos W y aplicando luego la función *Softmax*:

$$\hat{a} = Softmax(W(o+u)).$$

El modelo aprende las matrices A, B, C, W minimizando la pérdida de entropía cruzada estándar entre $\hat{a}$ y la verdadera respuesta a.

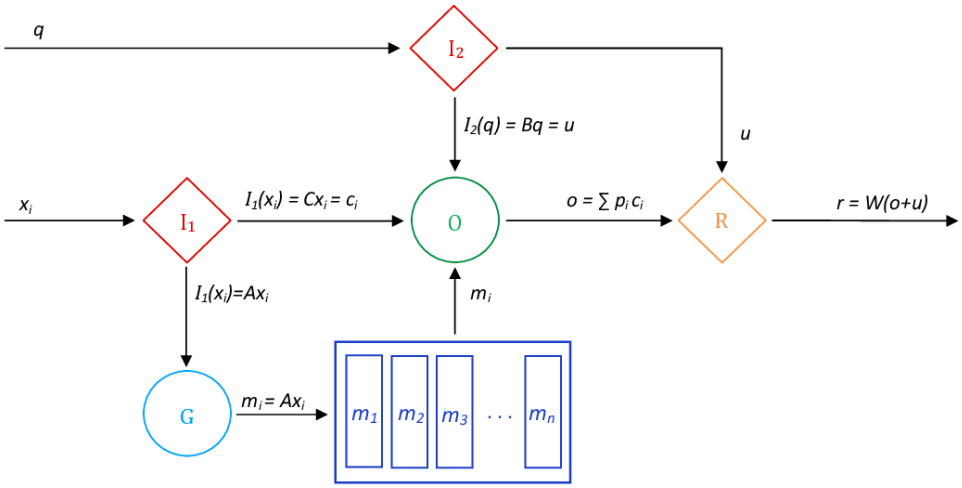


Figura 3.9: Esquema de una Red de Memoria End-to-End.

Representaciones:

Las frases (datos de entrada x_i) pueden representarse mediante el modelo *BOW*, como también la pregunta q:

$$m_i = \sum_j A x_{ij}, \qquad c_i = \sum_j C x_{ij}, \qquad u = \sum_j B q_j,$$

siendo $x_i = x_{i1}, \ldots, x_{in}$, x_{ij} cada una de las palabras que conforman la frase x_i, que se representan como un vector *one-hot*.

A diferencia de esta representación, que no tiene en cuenta el orden de las palabras en la frase, se considerará en la implementación del modelo la representación mediante codificación de posición, haciendo que el orden de las palabras afecte a los vectores memoria:

$$m_i = \sum_j l_j \cdot A x_{ij},$$

donde '$\cdot$' es una multiplicación elemento a elemento y l_j es el vector columna definido en 2.1.2.

Modificaciones:

Puede hacerse que el modelo realice múltiples saltos (*hops*) si se construye con varias capas en lugar de solo con una, como se muestra en la figura 3.10, pues la utilización de más de una capa da mejores resultados de acierto a la hora de predecir respuestas. El funcionamiento será el mismo, pero ahora se tendrán A^k y C^k matrices diferentes para codificar los datos de entrada para cada capa, con $k = 1, \ldots, K$ siendo K el número de capas. Las entradas a partir de la primera capa son la suma de la salida o^k y la entrada u^k de la capa k, es decir

$$u^{k+1} = o^k + u^k.$$

Tras haber recorrido todas las capas, la predicción final será

$$\hat{a} = \text{Softmax}(Wu^{K+1}) = \text{Softmax}(W(o^K + u^K)).$$

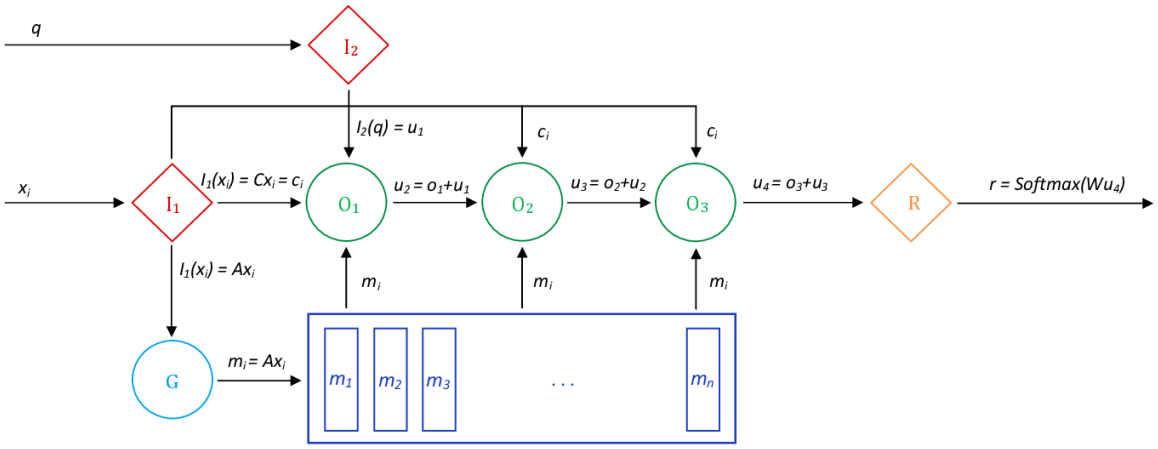


Figura 3.10: Esquema de una Red de Memoria End-to-End de tres capas, con $A^1 = A^2 = A^3 = A$ y $C^1 = C^2 = C^3 = C$.

Notar que si se considera que las matrices de inmersión a lo largo de las diferentes capas son iguales, es decir $A^1 = A^2 = \cdots = A^K$ y $C^1 = C^2 = \cdots = C^K$, este modelo puede verse como una red neuronal recurrente tradicional en la que hay dos tipos de salidas: una salida interna a partir de las inmersiones A^i que son las probabilidades p_i (pesos) y una salida externa que es la respuesta que se predice a partir de esas probabilidades y de las inmersiones C^i. En la figura 3.11 se ilustra esta visualización de la red de memoria *end-to-end* como una red neuronal recurrente de las introducidas en la sección 3.1.

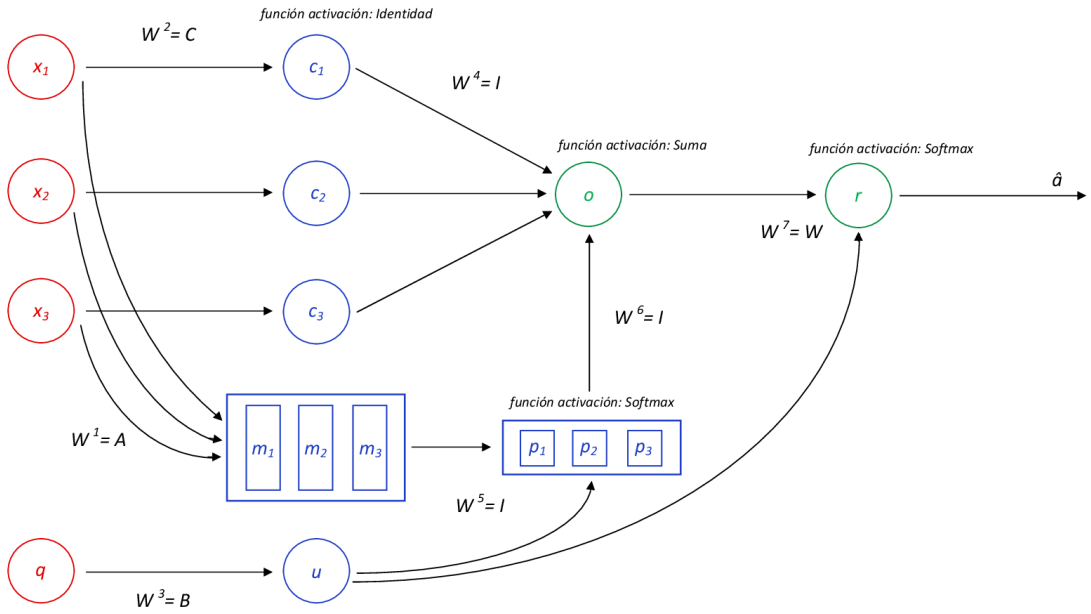


Figura 3.11: Esquema de una Red de Memoria End-to-End visualizada como una RNN.

Viendo la red de esta manera, es sencillo definir lo que serían las matrices de pesos y aplicar el método de retropropagación para la actualización de los pesos explicado en 3.1.1, como se muestra en la figura 3.12.

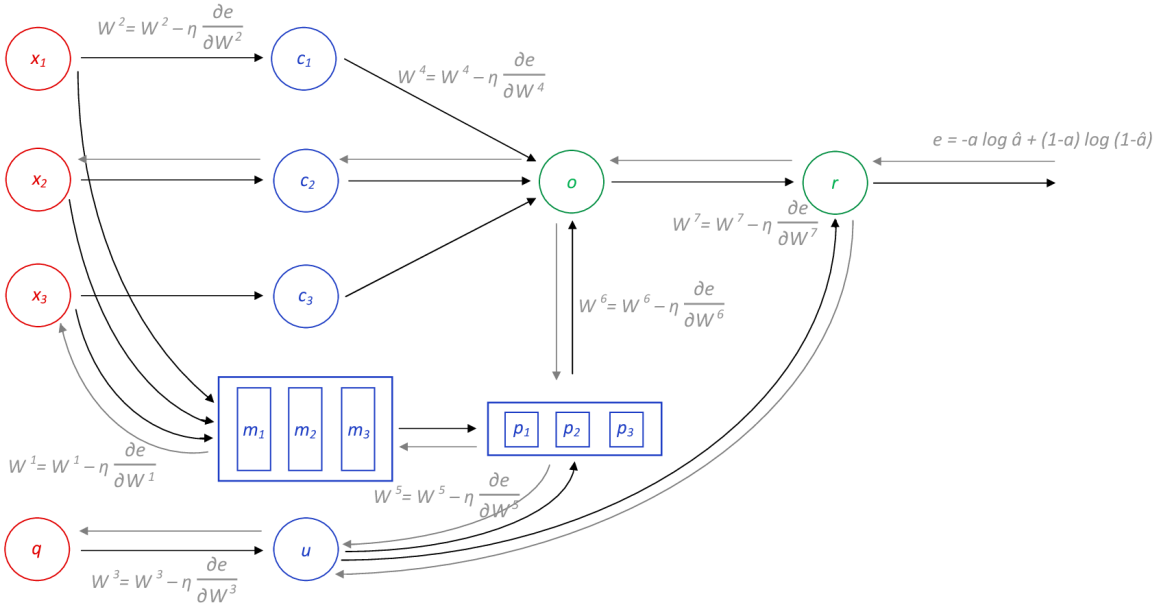


Figura 3.12: Actualización de los pesos mediante retropropagación.

Ejemplo:

A continuación se muestra una historia formada por $I = 4$ frases $\{x_i\}$, una pregunta q acerca de lo acontecido en esa situación y la respuesta a correcta. Sea x_{ij} la $j-$ésima palabra de la frase i, representada por un vector *one-hot* (vector con un 1 y lo demás 0) de longitud $V = 13$ (V es el tamaño del vocabulario y dado que en este ejemplo aparecen 13 palabras distintas, se considera que este es el tamaño total del vocabulario por simplificar). La misma representación es usada también para la pregunta y la respuesta.

```
Sam walks into the kitchen.
Sam picks up the apple.
Sam walks into the bedroom.
Sam drops the apple.
Where is the apple?      bedroom
```

Se tiene:

$x_1 =$	Sam	walks	into	the	kitchen
$x_2 =$	Sam	picks	up	the	apple
$x_3 =$	Sam	walks	into	the	bedroom
$x_4 =$	Sam	drops	the	apple	–
$q =$	Where	is	the	apple	–
$a =$	Bedroom	–	–	–	–

Entonces:

$$
\begin{aligned}
x_{11} &= \text{Sam} && = (1,0,0,0,0,0,0,0,0,0,0,0,0,0)\\
x_{12} &= \text{walks} && = (0,1,0,0,0,0,0,0,0,0,0,0,0,0)\\
x_{13} &= \text{into} && = (0,0,1,0,0,0,0,0,0,0,0,0,0,0)\\
x_{14} &= \text{the} && = (0,0,0,1,0,0,0,0,0,0,0,0,0,0)\\
x_{15} &= \text{kitchen} && = (0,0,0,0,1,0,0,0,0,0,0,0,0,0)
\end{aligned}
$$

$$
\begin{aligned}
x_{21} &= \text{Sam} && = (1,0,0,0,0,0,0,0,0,0,0,0,0,0)\\
x_{22} &= \text{picks} && = (0,0,0,0,0,1,0,0,0,0,0,0,0,0)\\
x_{23} &= \text{up} && = (0,0,0,0,0,0,1,0,0,0,0,0,0,0)\\
x_{24} &= \text{an} && = (0,0,0,0,0,0,0,1,0,0,0,0,0,0)\\
x_{25} &= \text{apple} && = (0,0,0,0,0,0,0,0,1,0,0,0,0,0)
\end{aligned}
$$

$$
\begin{aligned}
x_{31} &= \text{Sam} && = (1,0,0,0,0,0,0,0,0,0,0,0,0,0)\\
x_{32} &= \text{walks} && = (0,1,0,0,0,0,0,0,0,0,0,0,0,0)\\
x_{33} &= \text{into} && = (0,0,1,0,0,0,0,0,0,0,0,0,0,0)\\
x_{34} &= \text{the} && = (0,0,0,1,0,0,0,0,0,0,0,0,0,0)\\
x_{35} &= \text{bedroom} && = (0,0,0,0,0,0,0,0,0,0,1,0,0,0)
\end{aligned}
$$

$$
\begin{aligned}
x_{41} &= \text{Sam} && = (1,0,0,0,0,0,0,0,0,0,0,0,0,0)\\
x_{42} &= \text{drops} && = (0,0,0,0,0,0,0,0,0,0,0,1,0,0)\\
x_{43} &= \text{the} && = (0,0,0,1,0,0,0,0,0,0,0,0,0,0)\\
x_{44} &= \text{apple} && = (0,0,0,0,0,0,0,0,1,0,0,0,0,0)
\end{aligned}
$$

$$
\begin{aligned}
q_1 &= \text{Where} && = (0,0,0,0,0,0,0,0,0,0,0,0,1,0)\\
q_2 &= \text{is} && = (0,0,0,0,0,0,0,0,0,0,0,0,0,1)\\
q_3 &= \text{the} && = (0,0,0,1,0,0,0,0,0,0,0,0,0,0)\\
q_4 &= \text{apple} && = (0,0,0,0,0,0,0,0,1,0,0,0,0,0)
\end{aligned}
$$

$$
a = \text{Bedroom} \quad = (0,0,0,0,0,0,0,0,0,1,0,0,0,0)
$$

Para codificar las frases y la pregunta, se utiliza el método *BOW*, y se consideran las matrices A, B y C iguales. Estas matrices se inicializan aleatoriamente y el modelo las va modificando y aprendiendo conforme se entrena, pero como solo se está considerando un ejemplo pequeño para comprender su funcionamiento, se eligen de esta manera.

***BOW* = [Sam, walks, into, the, kitchen, picks, up, an, apple, bedroom, drops, Where, is]**

Se cuenta el número de veces que aparece una palabra de la bolsa de palabras en cada uno de los vectores. En este ejemplo no hay palabras que se repitan en la misma frase, pero si se tuviera que en x_1 'Sam' aparece dos veces, en lugar de asignarle 1 a la palabra 'Sam' se le asignaría 2.

$$
\begin{aligned}
x_1 &\quad (1,1,1,1,1,0,0,0,0,0,0,0,0)\\
x_2 &\quad (1,0,0,0,0,1,1,1,1,0,0,0,0)\\
x_3 &\quad (1,1,1,1,0,0,0,0,0,1,0,0,0)\\
x_4 &\quad (1,0,0,1,0,0,0,0,1,0,0,1,1)\\
q &\quad (0,0,0,1,0,0,0,0,1,0,0,1,1)
\end{aligned}
$$

Considérese por ejemplo la siguiente matriz de inmersión:

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix}.$$

A continuación se van a calcular los diferentes vectores de memoria $m_i = \sum_j Ax_{ij}$:

$$Ax_{11} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

y los demás de igual modo, por lo que se obtienen los siguientes vectores memoria:

$$m_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 5 \\ 1 \\ 4 \\ 2 \\ 1 \end{pmatrix}$$

$$m_2 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 5 \\ 1 \\ 2 \\ 1 \end{pmatrix}$$

$$m_3 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \\ 5 \\ 2 \\ 1 \end{pmatrix}$$

$$m_4 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ 2 \\ 4 \\ 2 \end{pmatrix}$$

También se realiza la codificación de la pregunta: $u = \sum_j Bq_j$ (recordar que se consideraba $B = A$), luego

$$u = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \\ 4 \end{pmatrix}$$

Ahora tienen que calcularse las probabilidades correspondientes a cada memoria al compararlas con la pregunta: $p_i = \text{Softmax}(u^T m_i)$.

$$u^T m_1 = \begin{pmatrix} 1 & 1 & 1 & 2 & 4 \end{pmatrix} \begin{pmatrix} 5 \\ 1 \\ 4 \\ 2 \\ 1 \end{pmatrix} = 18, \quad u^T m_2 = 15, \quad u^T m_3 = 158, \quad u^T m_4 = 22$$

luego

$$p_1 = Softmax(u^T m_1) = \frac{e^{18}}{e^{18} + e^{15} + e^{18} + e^{22}} = 0{,}01765$$

$$p_2 = Softmax(u^T m_2) = \frac{e^{15}}{e^{18} + e^{15} + e^{18} + e^{22}} = 0{,}00088$$

$$p_3 = Softmax(u^T m_3) = \frac{e^{18}}{e^{18} + e^{15} + e^{18} + e^{22}} = 0{,}01765$$

$$p_4 = Softmax(u^T m_4) = \frac{e^{22}}{e^{18} + e^{15} + e^{18} + e^{22}} = 0{,}96382$$

Una vez calculadas las probabilidades, se calcularían los vectores de salida (que en este caso no es necesario ya que al considerar la matriz C igual a la A son los mismos que los m_i) y después el vector $o = \sum_i p_i c_i$.

$$o = 0{,}01765 \begin{pmatrix} 5 \\ 1 \\ 4 \\ 2 \\ 1 \end{pmatrix} + 0{,}00088 \begin{pmatrix} 1 \\ 5 \\ 1 \\ 2 \\ 1 \end{pmatrix} + 0{,}01675 \begin{pmatrix} 4 \\ 1 \\ 5 \\ 2 \\ 1 \end{pmatrix} + 0{,}96382 \begin{pmatrix} 2 \\ 2 \\ 2 \\ 4 \\ 2 \end{pmatrix} = \begin{pmatrix} 2{,}08737 \\ 2{,}03794 \\ 2{,}08737 \\ 3{,}92776 \\ 1{,}96382 \end{pmatrix}$$

Por último, se tiene que sumar este vector o con el vector u, y multiplicarlo por una matriz de pesos W que también va aprendiendo el modelo tratando de minimizar el error entre la respuesta predicha $\hat{a}$ y la respuesta real a. Para este ejemplo también se necesita fijarla, por lo que se ha considerado también igual a A. Así:

$$W(o+u) = W \begin{pmatrix} 3{,}08737 \\ 3{,}03794 \\ 3{,}08737 \\ 5{,}92776 \\ 5{,}96382 \end{pmatrix} = \begin{pmatrix} 5{,}96382 \\ 5{,}96382 \\ 5{,}92776 \\ 3{,}08737 \\ 14{,}92952 \\ 3{,}03794 \\ 3{,}03794 \\ 3{,}03794 \\ 3{,}08737 \\ 18{,}06632 \\ 6{,}17474 \\ 6{,}17474 \\ 015{,}14044 \end{pmatrix}$$

y entonces

$$\hat{a} = Softmax(W(o+u)) = \begin{pmatrix} 5{,}05494 \cdot 10^{-6} \\ 5{,}05494 \cdot 10^{-6} \\ 4{,}87591 \cdot 10^{-6} \\ 2{,}84767 \cdot 10^{-7} \\ 0{,}03958 \\ 2{,}71033 \cdot 10^{-7} \\ 2{,}71033 \cdot 10^{-7} \\ 2{,}71033 \cdot 10^{-7} \\ 2{,}84767 \cdot 10^{-7} \\ \mathbf{0{,}911518} \\ 6{,}24191 \cdot 10^{-6} \\ 6{,}24191 \cdot 10^{-6} \\ 0{,}04887 \end{pmatrix}, \quad \text{y se tenía que } a = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

luego el modelo predice con un acierto del 91,15 % que la respuesta a la consulta realizada es *'bedroom'*, lo que es correcto.

3.2.3. Red de Memoria Clave-Valor (KV-MemNN)

Este modelo está basado en la arquitectura del anterior, es decir está entrenado *end-to-end*, y se diferencia en que presenta dos componentes de memoria distintas, denominadas clave y valor, con la idea de facilitar la búsqueda de información relevante con respecto a la pregunta.

Presenta una fase de direccionamiento y lectura de la memoria, y una segunda fase de acceso a la misma. La fase de direccionamiento se basa en la memoria clave, almacenando la información relevante con respecto a la pregunta realizada, mientras que la fase de lectura (que devuelve el resultado) utiliza la memoria valor. El esquema de su funcionamiento se detalla a continuación y se muestra en la figura 3.13, igualmente partiendo del esquema general de la red de memoria.

Sea x la pregunta en cuestión y $(k_1, v_1), \ldots, (k_M, v_M)$ las parejas de vectores de memoria sobre la información disponible. El direccionamiento y lectura de la memoria consta de tres pasos:

1. En el artículo original [7] se define como primer paso la utilización de la pregunta para preseleccionar un subconjunto de memorias de tamaño N $(k_{h_1}, v_{h_1}), \ldots, (k_{h_N}, v_{h_N})$ cuyas claves compartan al menos una palabra con la pregunta con frecuencia <1000 para deshacernos de las palabras vacías[1] (*stop words*). En la implementación aportada en este trabajo se hace algo similar, que es utilizar la pregunta para preseleccionar los subconjunto de información en los que aparecen las entidades que están presentes en estas preguntas, y entonces almacenarlas en las memorias (k_{h_i}, v_{h_i})). Este método es igualmente una búsqueda de palabras compartidas.

2. Se codifican los datos de entrada k_{h_i}, v_{h_i} y las preguntas x, se almacenan en las memorias correspondientes y se asignan probabilidades de relevancia a las memorias valor a partir de la comparación de las memorias clave con la pregunta:

$$p_{h_i} = \text{Softmax}(A\Phi_X(x) \cdot A\Phi_K(k_{h_i})),$$

   con $\Phi_\_$ funciones características de dimensión D, A una matriz $d \times D$.

3. En la etapa final de lectura, se devuelve el vector de la suma ponderada de los valores v_{h_i} de las memorias mediante sus probabilidades asociadas:

$$o = \sum_i p_{h_i} A\Phi_V(v_{h_i}).$$

Tras esto, se actualiza la consulta a partir de la salida obtenida o y la consulta previa $q = A\Phi_X(x)$, siendo la nueva consulta $q_2 = R_1(q+o)$ donde R_1 es una matriz $d \times d$. A continuación, tras este primer salto, se realizarán $H-1$ saltos más donde repetirán los pasos 2 y 3 así como el acceso a la memoria, considerando para cada salto

$$p_{h_i} = \text{Softmax}(q_{j+1}^T A\Phi_K(k_{h_i})), \qquad q_{j+1} = R_j(q_j + o).$$

Notar que la respuesta o también se actualiza con cada salto.

Estas actualizaciones permiten contar con consultas que posean información más relevante para los siguientes accesos. Tras los H saltos se calcula la predicción final $\hat{a}$ sobre todas las posibles salidas y_i:

$$\hat{a} = \underset{i=1,\ldots,C}{\text{argmax}}\ \text{Softmax}(q_{H+1}^T B\Phi_Y(y_i)),$$

donde y_i son los posibles candidatos a respuesta (todas las entidades almacenadas).

El modelo aprende a mejorar los accesos a la memoria para devolver el objetivo deseado a minimizando la pérdida de entropía cruzada entre $\hat{a}$ y la respuesta correcta a. Para aprender las matrices $A, B, R_1, \ldots, R_H$ se utilizan los métodos de retropropagación y gradiente descendente estocástico.

[1] artículos, pronombres, preposiciones, etc.

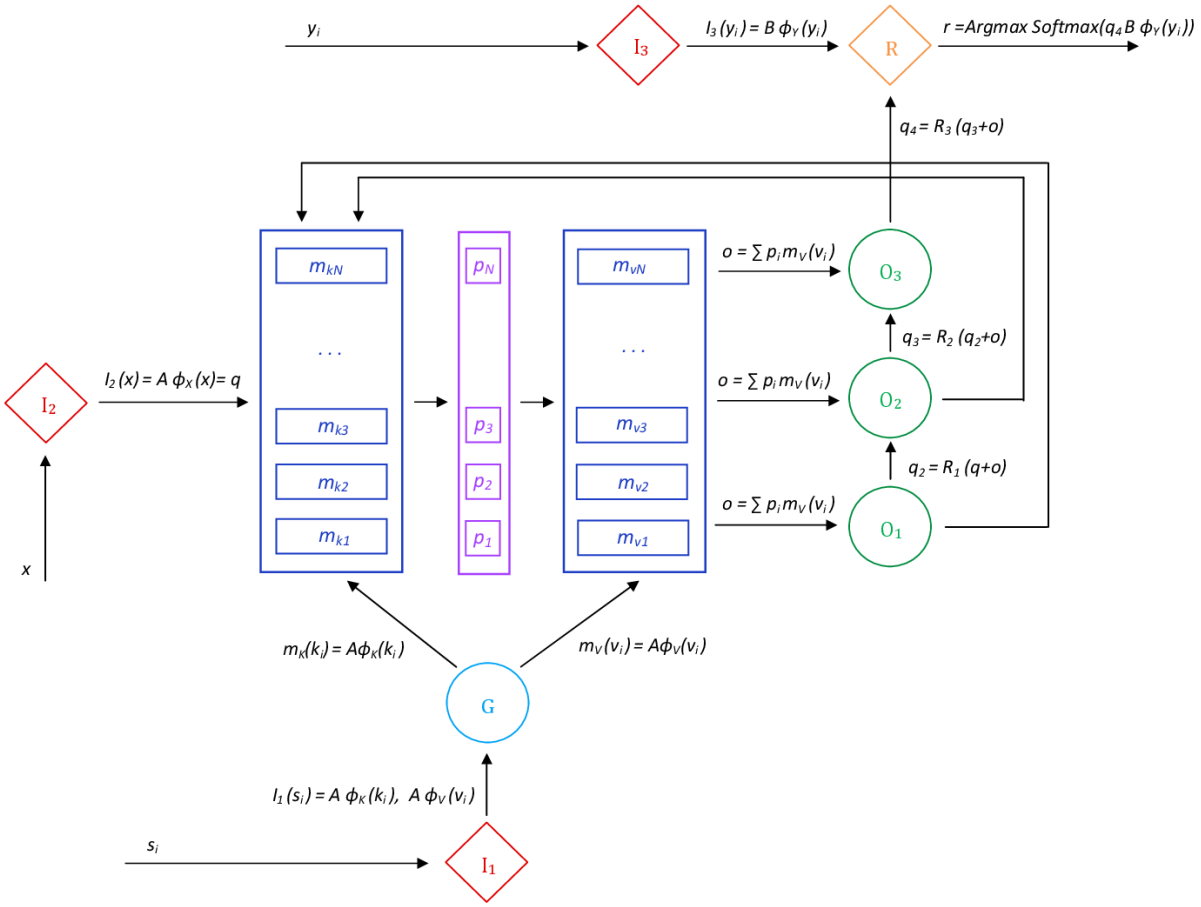


Figura 3.13: Esquema de una Red de Memoria Clave-Valor con 3 saltos.

Representaciones:

Como se ha dicho, lo que hace potente a este modelo es la habilidad de representar y codificar el conocimiento en memoria clave y memoria valor, que son distintas pero están relacionadas. Podemos elegir el tipo de codificación a la hora de definir las funciones Φ_X, Φ_Y, Φ_K y Φ_V para la pregunta, la respuesta, las claves y los valores, respectivamente.

Las representaciones que se han contemplado son los formatos frase, ventana y tripleta comentados en 2.4.2, y como método de codificación el *BOW*.

Ejemplo:

Para este ejemplo se considera el conjunto de datos *WikiMovies*. Suponer que está formado por una sola historia, es decir solo se dispone de información de una sola película. La representación considerada de dicha información es el formato de tripletas para simplificar las explicaciones. Además se tiene que el tamaño de las características es $d = 12$, el del espacio de inmersión $D = 20$ y $H = 3$ saltos.

```
1 Flags of Our Fathers directed_by Clint Eastwood
2 Flags of Our Fathers written_by Paul Haggis, William Broyles Jr., Ron Powers,
    James Bradley
```

```
3 Flags of Our Fathers starred_actors Ryan Phillippe, Jesse Bradford, Adam Beach,
     John Benjamin Hickey
4 Flags of Our Fathers release_year 2006
5 Flags of Our Fathers in_language English
6 Flags of Our Fathers has_genre Drama, War, History
7 Flags of Our Fathers has_imdb_rating good
8 Flags of Our Fathers has_imdb_votes famous
```

Además de la información sobre la película, el conjunto de entrenamiento cuenta con 13 preguntas referentes a ella, que se irán seleccionando de una en una para el entrenamiento (suponiendo que se considera un tamaño de batch 1). Se mostrará aquí tan sólo una pregunta por simplificación.

```
1 the director of Flags of Our Fathers was who? Clint Eastwood
```

Así, siguiendo el esquema de la red en la figura 3.13:

$s_1 =$ Flags of Our Fathers directed_by Clint Eastwood
$s_2 =$ Flags of Our Fathers written_by Paul Haggis, William Broyles Jr., ...
$s_3 =$ Flags of Our Fathers starred_actors Ryan Phillippe, Jesse Bradford, ...
$s_4 =$ Flags of Our Fathers release_year 2006
$s_5 =$ Flags of Our Fathers in_language English
$s_6 =$ Flags of Our Fathers has_genre Drama, War, History
$s_7 =$ Flags of Our Fathers has_imdb_rating good
$s_8 =$ Flags of Our Fathers has_imdb_votes famous

$x =$ the director of Flags of Our Fathers was who
$a =$ Clint Eastwood

$y_1 =$	Flags of Our Fathers	$y_6 =$	History	$y_{11} =$	Ron Powers
$y_2 =$	2006	$y_7 =$	film	$y_{12} =$	1945
$y_3 =$	American	$y_8 =$	William Broyles	$y_{13} =$	battle
$y_4 =$	War	$y_9 =$	Paul Haggis	$y_{14} =$	flag
$y_5 =$	Drama	$y_{10} =$	James Bradley	$y_{15} =$	iwo jima

siendo y_i con $i = 1, \ldots, 15$ las entidades consideradas para este conjunto de datos y que son las posibles respuestas que dará el modelo.

Como se explicó en la subsección 2.4.2, el sujeto y la relación de la tripeta se consideran como la parte clave y el objeto como la parte valor. Es decir:

$k_1 =$	Flags of Our Fathers directed_by	$v_1 =$	Clint Eastwood
$k_2 =$	Flags of Our Fathers written_by	$v_2 =$	Paul Haggis, William Broyles Jr., ...
$k_3 =$	Flags of Our Fathers starred_actors	$v_3 =$	Ryan Phillippe, Jesse Bradford, ...
$k_4 =$	Flags of Our Fathers release_year	$v_4 =$	2006
$k_5 =$	Flags of Our Fathers in_language	$v_5 =$	English
$k_6 =$	Flags of Our Fathers has_genre	$v_6 =$	Drama, War, History
$k_7 =$	Flags of Our Fathers has_imdb_rating	$v_7 =$	good
$k_8 =$	Flags of Our Fathers has_imdb_votes	$v_8 =$	famous

A continuación, se carga y analiza la información sobre la película y la pregunta tal y como se explicó en la subsección 2.4.2. Se crea un diccionario con todas las palabras presentes en la historia, pregunta, respuesta y listado de entidades.

Una vez hecho esto, se vectorizan k_i, v_i y x, en este caso reemplazando las palabras por su índice en el diccionario ($k_1 = [16 \quad 74]$, $v_1 = [62]$, $x = [104 \quad 57 \quad 84 \quad 16 \quad 109 \quad 112]$). Estos vectores tendrán que rellenarse con ceros hasta alcanzar el tamaño máximo de frase en todo el conjunto de datos, que en este caso es 10. Así, estos vectores se guardarán en tensores de dimensión (13, 8, 10) en el caso de las claves y valores, (13, 10) en el caso de las preguntas y (13, 124) para las respuestas. En el caso de la respuesta a y las candidatas a respuesta se representan mediante vector one-hot. Estas últimas se guardarán en un tensor de dimensiones (15, 124).

Cuando ya se tienen todos los datos en el formato adecuado, se procede a aplicar el modelo. Se selecciona un conjunto de preguntas, en este caso una dado que el tamaño del batch seleccionado es 1, y sus historias asociadas (ya separadas en parte clave y valor). Es decir, se tienen los siguientes tensores:

- x de dimensiones (1, 10)
- k_i de dimensiones (1, 8, 10)
- v_i de dimensiones (1, 8, 10)

que se transformarán mediante la función característica $\Phi\_$ de dimensión $D = 20$, teniendo:

- $\Phi_X(x)$ de dimensiones (1, 10, 20)
- $\Phi_K(k_i)$ de dimensiones (1, 8, 10, 20)
- $\Phi_V(v_i)$ de dimensiones (1, 8, 10, 20)

Después se realizan una serie de operaciones internas para reducir estas dimensiones y poder multiplicar los tensores por la matriz $A_{(12\times20)}$ y convertirlos al mismo espacio de inmersión. Así, se tendrá

- $\Phi_X(x)$ de dimensiones (1, 20), luego $A\Phi_X(x)$ de dimensiones (1,1,12)
- $\Phi_K(k_i)$ de dimensiones (1, 8, 20), luego $A\Phi_K(k_i)$ de dimensiones (1, 8, 12)
- $\Phi_V(v_i)$ de dimensiones (1, 8, 20), luego $A\Phi_V(v_i)$ de dimensiones (1, 8, 12)

Lo realizado hasta aquí se correspondería con el paso de los datos por las componentes I_1 e I_2. Una vez se tienen los vectores transformados, se realiza el primer salto a través del modelo, almacenando en la memoria los tensores correspondientes a la clave y al valor, es decir $m_{k_i} = A\Phi_K(k_i)$ y $m_{v_i} = A\Phi_V(v_i)$ mediante la actualización que realiza la componente de generalización G. Luego se multiplica la pregunta $q = A\Phi_X(x)$ por la información guardada en las memorias m_{k_i} y se aplica la función *Softmax*, teniendo así un tensor de dimensiones (1, 8, 1) que contiene las probabilidades p_i de relevancia de cada memoria para responder a la pregunta considerada.

Ahora se multiplican estas probabilidades por lo almacenado en las memorias valor, generando la salida $o = \sum_i p_i m_{v_i}$, tensor de dimensiones (1, 12). Esto, sumado al tensor q, a través de la componente de salida O_1 mediante la matriz $R_{1(12\times12)}$ se transformaría en la siguiente consulta $q_2 = R_1(q+o)$, que

volvería a compararse con las memorias clave m_{k_i} y obtener así unas nuevas probabilidades de relevancia. Este procedimiento se realizará durante el número de saltos indicado hasta obtener la consulta final $q_4 = R_3(q_3 + o)$, que tiene dimensiones (12, 1).

De igual manera que se había hecho con las preguntas, claves y objetos, la componente I_3 transforma el tensor de respuestas candidatas mediante una función característica Φ_Y también de dimensión $D = 20$ y lo convierte luego al espacio de inmersión mediante una matriz $B_{(12\times 20)}$ (que puede ser la misma matriz que A), quedando un tensor $r_i = B\Phi_Y(y_i)$ de dimensiones (12, 124). Tras esto se multiplicará el tensor de candidatas por la última consulta generada y se aplicará de nuevo la función *Softmax* para asignar a cada candidata la probabilidad que tiene de ser elegida como respuesta predicha: $\hat{p}_i = Softmax(q_4 r_i)$.

Finalmente, en la componente R, se aplica la función *Argmax* para seleccionar la mayor de las probabilidades $\hat{p}_i$ y generar la respuesta r correspondiente. Además, se calculará la pérdida de entropía cruzada entre la respuesta correcta y la predicha, para actualizar los pesos, es decir las matrices A, B y R_i mediante el método *Adam*.

Todo esto se realizará para cada batch en cada época, y se calculará el *accuracy* de la predicción como se explicó en el capítulo 1. Cuando se haya alcanzado el número máximo de épocas se validará el modelo con otro conjunto de datos distinto al utilizado para el entrenamiento. También puede realizarse la validación tras el entrenamiento en cada época, lo que permite tener mayor control sobre los pasos que va realizando el modelo y evitar el sobreentrenamiento. Esto es lo que se ha hecho para los entrenamientos analizados en el capítulo 4.

Capítulo 4

Análisis y optimización del entrenamiento de las redes

> *« La verdadera optimización es la contribución revolucionaria de la investigación moderna a los procesos de decisiones. »* — *George Dantzig*

El trabajo realizado se ha centrado, además de en el estudio teórico de la aplicación de redes neuronales y de memoria a la minería de textos, en el análisis del funcionamiento de estas para diferentes selecciones de parámetros, y controlando en cada momento el ajuste de los modelos tanto en el entrenamiento como en la posterior evaluación. Como se verá a continuación, variar los parámetros en los diferentes entrenamientos es importante, puesto que aparecen diferencias claras de unos a otros y se consigue mejorar el rendimiento de los modelos. Para los tres modelos se ha generado una aplicación que extrae los resultados del criterio de ajuste seleccionado, el *accuracy* en este caso, para los conjuntos de entrenamiento y validación de todos ellos, en cada una de las épocas. Se ha procedido a evaluar la red tras cada época porque es importante tener control sobre entrenamiento de la red en cada momento, en lugar de observar tan sólo el resultado final, para así evitar el sobreentrenamiento. Se han guardado todos los datos en ficheros `.csv` para después importarlos en `R` y hacer una representación gráfica de las salidas que ayuden a visualizar e interpretar los resultados. El código correspondiente a la creación de las gráficas se adjunta en el anexo B.

Aunque se ha estudiado la influencia de los parámetros para todo el conjunto de tareas, en la mayoría de los análisis se representa sólo una selección de las mismas para facilitar la visualización.

En las siguientes secciones se presentan diferentes resultados obtenidos en la evaluación del modelo sobre conjunto test, en función de la variación de los parámetros introducidos en la sección 3.2.1 para cada uno de los modelos: *RNN*, *MemN2N* y *KV-MemNN*, sobre el conjunto de datos *tareas bAbI*, sobre las 20 tareas. A partir de dichos resultados se hará una selección de parámetros óptima para cada modelo, y se presentará una tabla que muestra el acierto para cada tarea evaluando el modelo entrenado con dichos parámetros óptimos. Para concluir estos análisis se hará una comparación de los tres modelos.

Además de esto, existe una última sección que expone un análisis complementario. Aquí se observa que para el buen funcionamiento de los modelos, no sólo influye la correcta elección de los parámetros, sino también la disposición de la información y su representación previa al entrenamiento. Se comparan los principales representaciones introducidas en el segundo capítulo en la sección 2.4.2.

4.1. Resultados RNN para tareas bAbI

Tanto esta sección como las siguientes presentan la siguiente estructura. Se presenta una tabla que muestra el valor de todos los parámetros establecidos para el entrenamiento de la red en cuestión y los diferentes valores que se han considerado del parámetro que se está estudiando en cada momento. Posteriormente, se analizan los resultados obtenidos a partir de esos entrenamientos y se muestran en un conjunto de gráficas. En la mayoría de los casos, estas gráficas mostrarán los resultados en función de un único parámetro, pero en alguna ocasión los mostrarán en función de dos parámetros.

Tras finalizar el análisis de todos los parámetros y haber seleccionado el conjunto óptimo, se presenta una tabla comparando los tiempos de procesamiento y entrenamiento requeridos por el modelo para la obtención de los resultados. El tiempo de procesamiento abarca la lectura y análisis de la información, la transformación a datos numéricos y su codificación para convertirlos en los datos de entrada de la red. El tiempo de entrenamiento hace referencia desde que se inicia la primera época hasta la última. Al final de cada sección se presenta otra tabla con los resultados del *accuracy* tras evaluar los modelos en el conjunto test.

Épocas

Parámetros del entrenamiento	
Número de épocas	60 - 100 - 200
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.01 - 0.001
Tamaño de inmersión	50
Tamaño de batch	32
Tipo de celda	LSTM

Tabla 4.1: Parámetros seleccionados para el análisis del número de épocas.

Para seleccionar el número de épocas se ha entrenado la red durante 60, 100 y 200 épocas. Se han hecho dos estudios por separado, mostrados en las figuras 4.1 y 4.2, considerando tasas de aprendizaje 0.01 y 0.001. En ambos casos se requieren 200 épocas, ya que en varias tareas el acierto para 60 épocas, incluso para 100, es insuficiente. Por tanto los posteriores análisis se realizarán con el número de épocas fijado a 200.

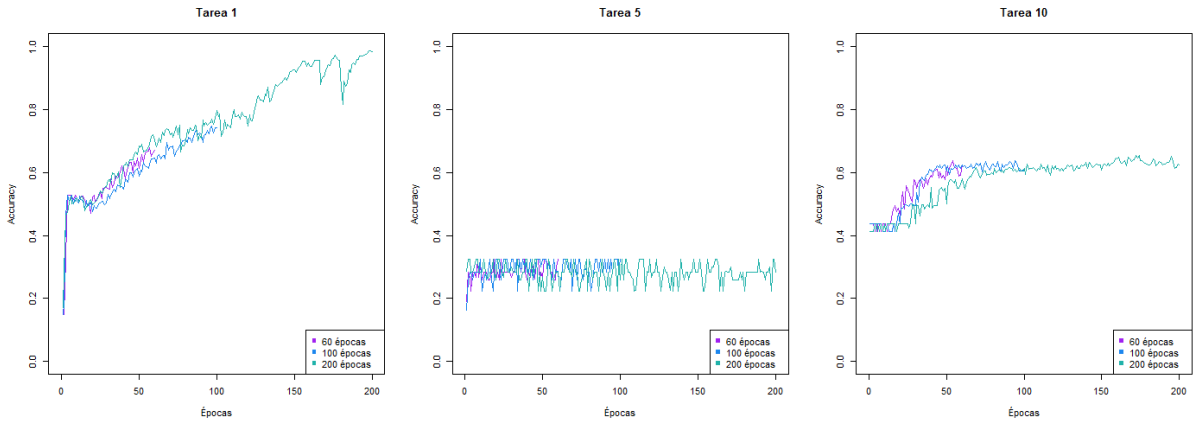


Figura 4.1: Resultados en función del número de épocas, con tasa de aprendizaje 0,01.

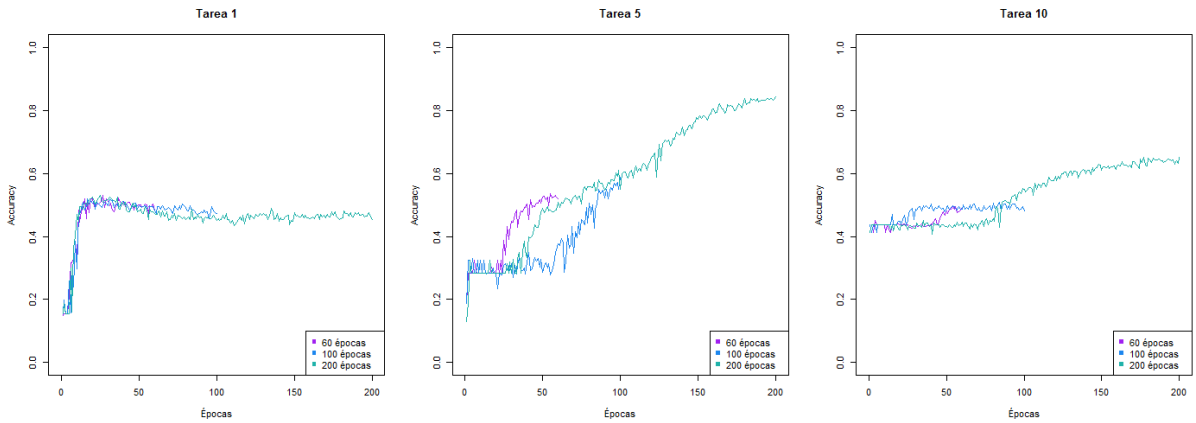


Figura 4.2: Resultados en función del número de épocas, con tasa de aprendizaje 0,001.

Tasa de aprendizaje

Se han considerado tres tasas de aprendizaje distintas: 0.1, 0.01 y 0.001, que implican un aprendizaje de la red de más a menos rápido. Las tres tasas se prueban para los diferentes algoritmos de optimización considerados.

Parámetros del entrenamiento	
Número de épocas	200
Algoritmo de optimización	Adam - SGD
Tasa de aprendizaje	0.1 - 0.01 - 0.001
Tamaño de inmersión	50
Tamaño de batch	32
Tipo de celda	LSTM

Tabla 4.2: Parámetros seleccionados para el análisis de la tasa de aprendizaje.

La figura 4.3 muestra los resultados obtenidos utilizando el algoritmo de optimización Adam. A pesar de la gráfica referente a la tarea 5, se seleccionará la tasa 0.01 como la más adecuada con la utilización de este algoritmo, ya que para el resto de tareas da mejores resultados.

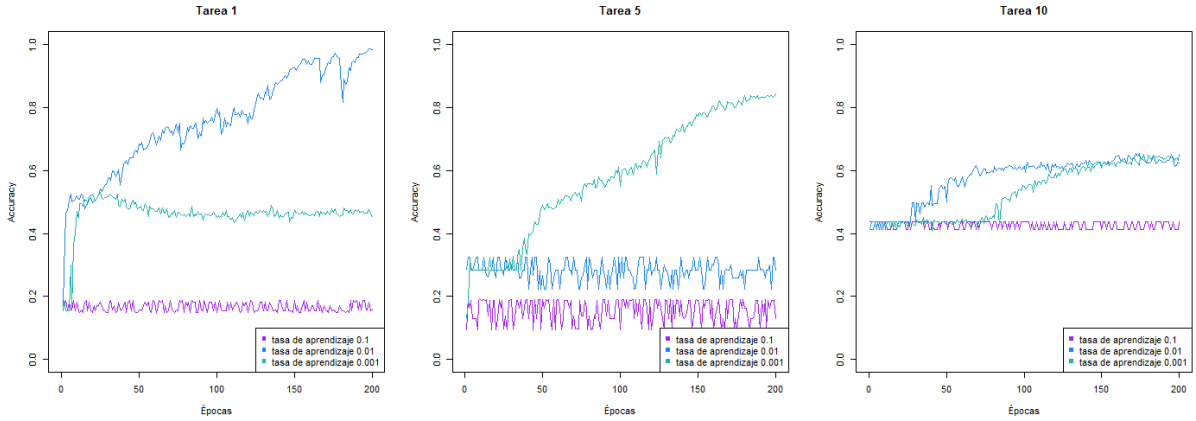


Figura 4.3: Resultados en función de la tasa de aprendizaje.

Por otro lado, de acuerdo con los resultados de la figura 4.4, en caso de utilizar como algoritmo de optimización el método del gradiente descendente estocástico, la tasa de aprendizaje apropiada sería 0.1, aunque los resultados obtenidos no alcanzan los anteriores.

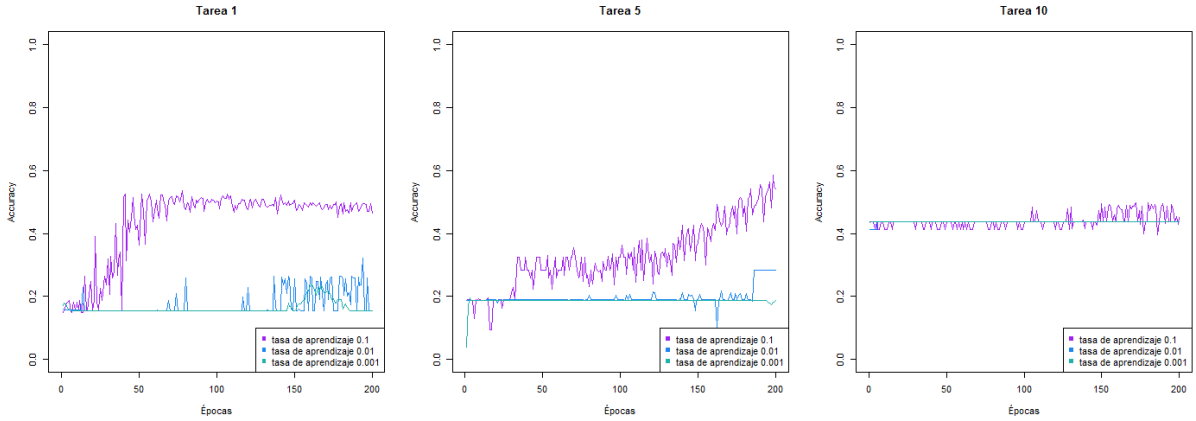


Figura 4.4: Resultados en función de la tasa de aprendizaje.

Algoritmo de optimización

Parámetros del entrenamiento	
Número de épocas	200
Algoritmo de optimización	Adam - SGD
Tasa de aprendizaje	Adam 0.01 - SGD 0.1
Tamaño de inmersión	50
Tamaño de batch	32
Tipo de celda	LSTM

Tabla 4.3: Parámetros seleccionados para el análisis del algoritmo de optimización.

A continuación se comparan directamente los dos algoritmos mencionados antes, y a la vista de los resultados previos, el método Adam utiliza la tasa de aprendizaje 0.01 y el gradiente descendente estocástico la 0.1.

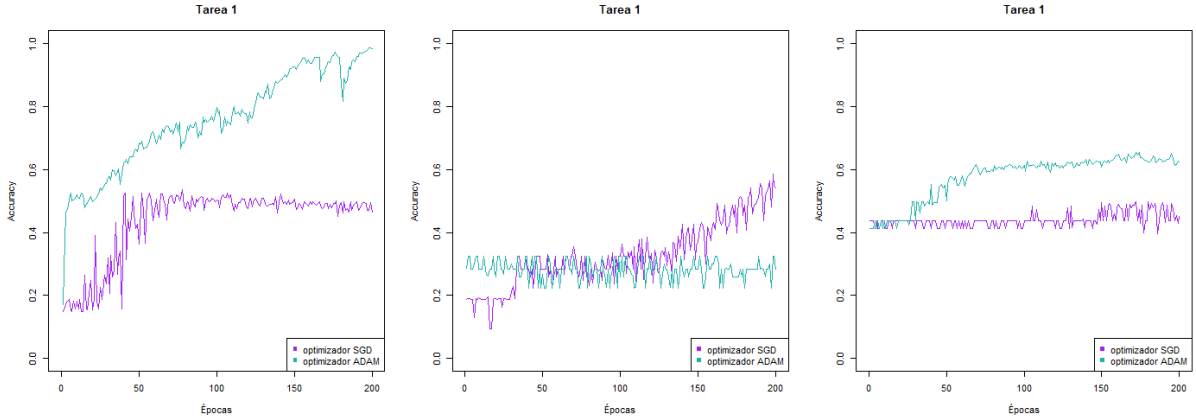


Figura 4.5: Resultados en función del algoritmo de optimización.

A la vista de los resultados de la figura 4.5 se seleccionará como método más conveniente el Adam, a pesar de que para la tarea 5, con la selección de parámetros considerada, sería mejor el SGD. Este resultado mejorará conforme vayan variándose el resto de parámetros.

Tipo de neurona

Parámetros del entrenamiento	
Número de épocas	200
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.01
Tamaño de inmersión	50
Tamaño de batch	32
Tipo de celda	LSTM - GRU - RNN simple

Tabla 4.4: Parámetros seleccionados para el análisis del tipo de celdas.

Como se comentó en el capítulo anterior, las redes recurrentes, además de su estructura básica, pueden ampliar su capacidad de aprendizaje considerando celdas de memoria en lugar de lo que serían las neuronas básicas.

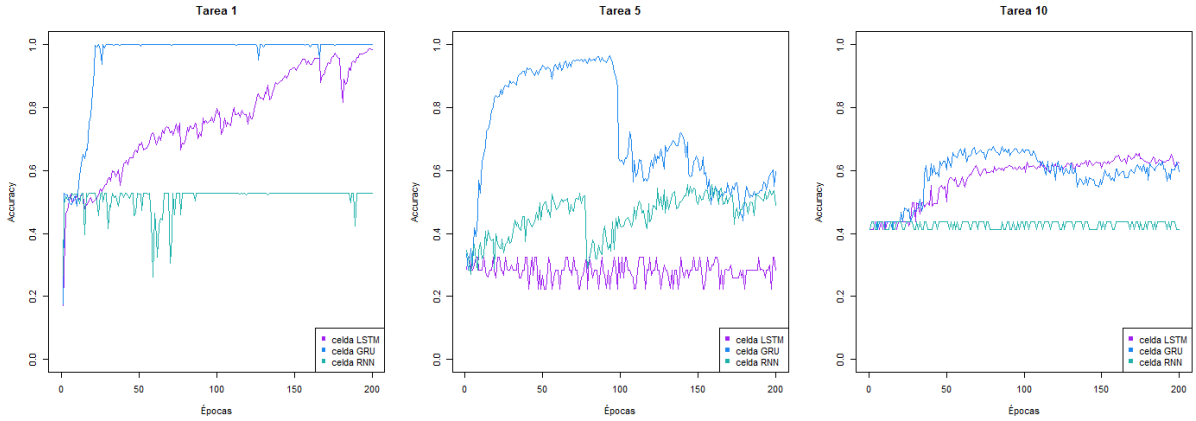


Figura 4.6: Resultados en función del tipo de celda.

Prestando atención a la figura 4.6, es conveniente considerar las celdas que cuentan con una cierta capacidad de memoria. De acuerdo con las tareas 1 y 10 la velocidad de convergencia es distinta, aunque el resultado óptimo es prácticamente el mismo. Considerando la GRU las redes se entrenan más rápido, pero puede producirse sobreentrenamiento (como ocurre con la tarea 5). Por otro lado, si se considera la LSTM, se alcanzan los mismos resultados aunque necesite más tiempo. No obstante, cuando se hayan terminado de elegir los valores óptimos para los parámetros restantes, se compararán de nuevo estas dos estructuras para seleccionar la más adecuada.

Tamaño de inmersión o embedding

Recordar que este parámetro establece la dimensión del espacio al que se transforma la información, y por tanto el número de neuronas que componen la capa oculta.

Parámetros del entrenamiento	
Número de épocas	200
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.01
Tamaño de inmersión	20 - 50 - 80
Tamaño de batch	32
Tipo de celda	LSTM

Tabla 4.5: Parámetros seleccionados para el análisis del tamaño de inmersión.

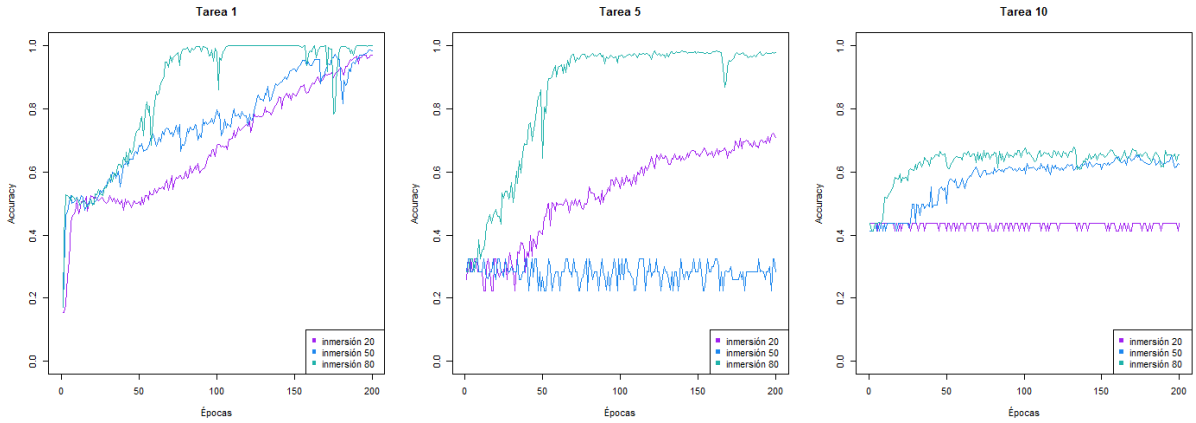


Figura 4.7: Resultados en función del tamaño de la inmersión.

La figura 4.7 muestra claramente que a mayor tamaño de inmersión, mayor es el acierto. La tarea 5 es la que presenta mayor mejoría, debido a que era con la que la red tenía problemas para conseguir buenos resultados.

Tamaño de grupo o batch

Parámetros del entrenamiento	
Número de épocas	200
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.01
Tamaño de inmersión	50 - 80
Tamaño de batch	10 - 32 - 50
Tipo de celda	LSTM

Tabla 4.6: Parámetros seleccionados para el análisis del tamaño del batch.

El estudio para la mejor selección de este parámetro se hará en función del tamaño de inmersión, considerando los casos 50 y 80, ya que en la figura 4.7 se veía que ambos tamaños proporcionan generalmente resultados parecidos, pero pueden crearse diferencias en función del batch que se considere.

Según los gráficos mostrados en la figura 4.8, si se considera 50 como dimensión del espacio de inmersión, el tamaño de batch más adecuado es diferente para cada tarea, pero se seleccionaría el batch 32 puesto que para la tarea 10 se obtienen mejores resultados y en el caso de la tarea 1 se alcanza el

mismo resultado tras las 200 épocas que considerando el batch 10, aunque con una convergencia más lenta. Observando ahora la figura 4.9 que es la que muestra los resultados de considerar un tamaño de inmersión 80, en este caso puede apreciarse que para todas las tareas la elección adecuada del batch es 32.

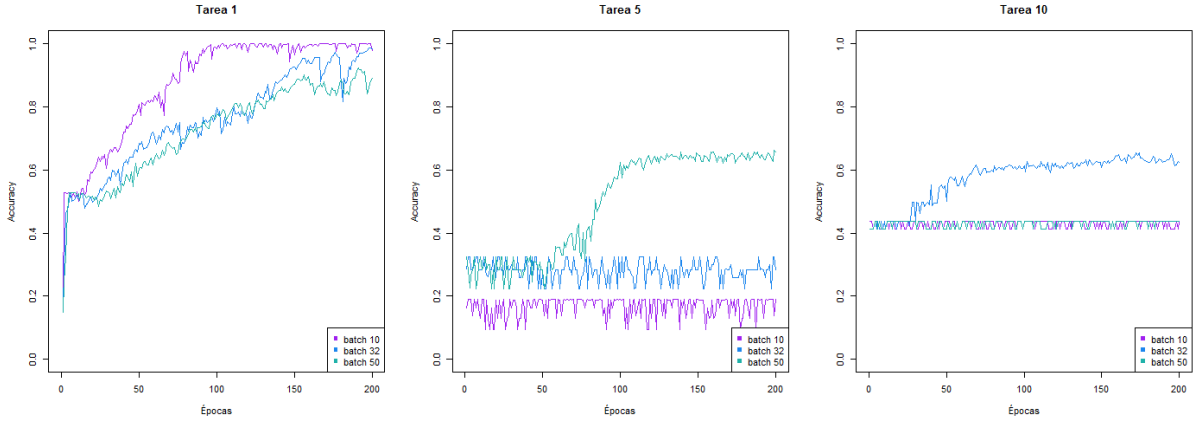


Figura 4.8: Resultados en función del tamaño del batch, con inmersión 50.

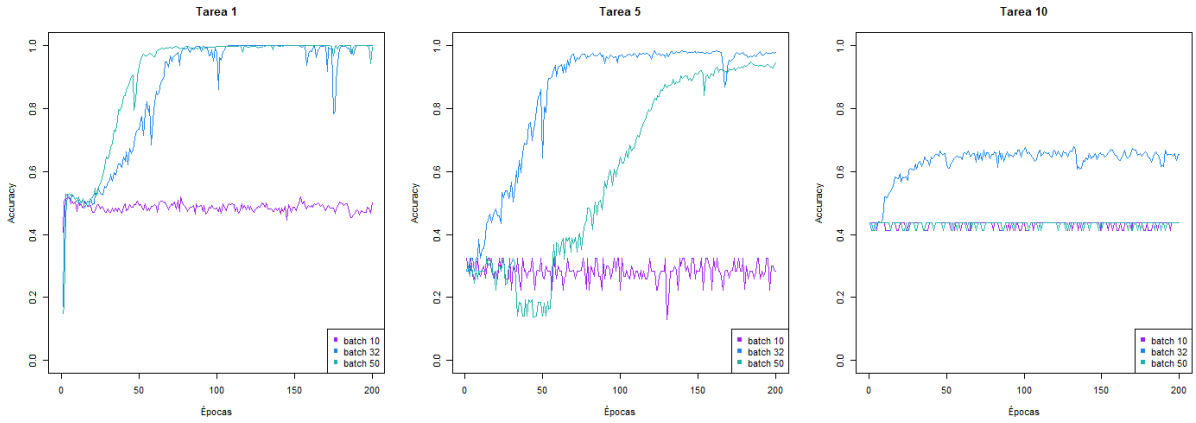


Figura 4.9: Resultados en función del tamaño del batch, con inmersión 80.

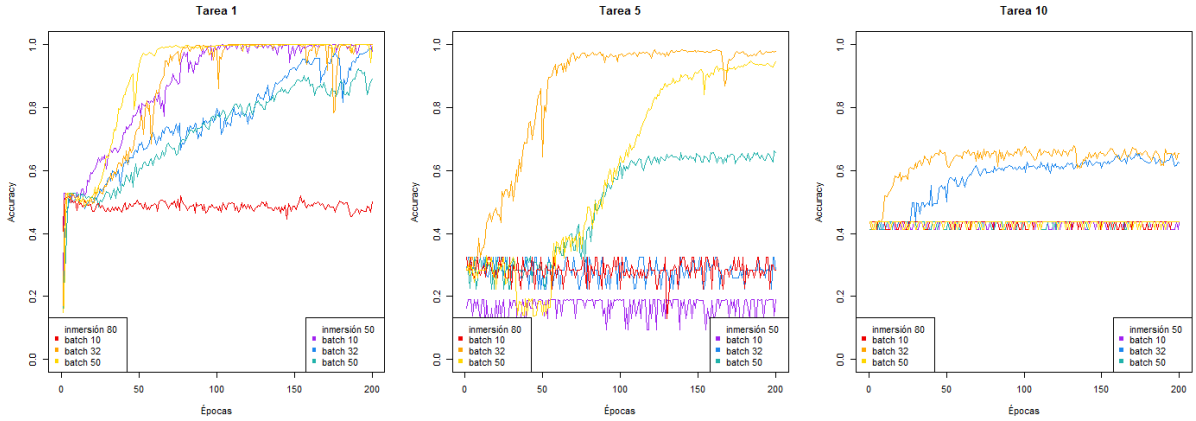


Figura 4.10: Resultados en función del tamaño del batch, comparando según inmersión 50 y 80.

Con la finalidad de comparar las dos opciones contempladas, se han unido las gráficas de las figuras anteriores en la figura 4.10. Se observa que el mejor de los resultados, y además para todas las tareas, se obtiene al considerar un tamaño de inmersión 80 y un tamaño de batch 32.

Por último, una vez se han fijado ya todos los parámetros del entrenamiento, se ha realizado una comparación para seleccionar el tipo de celda más adecuada, mostrada en la figura 4.11. Los resultados aquí mostrados reafirman la elección de LSTM como estructura óptima ya que con la otra estructura se produce sobreentrenamiento.

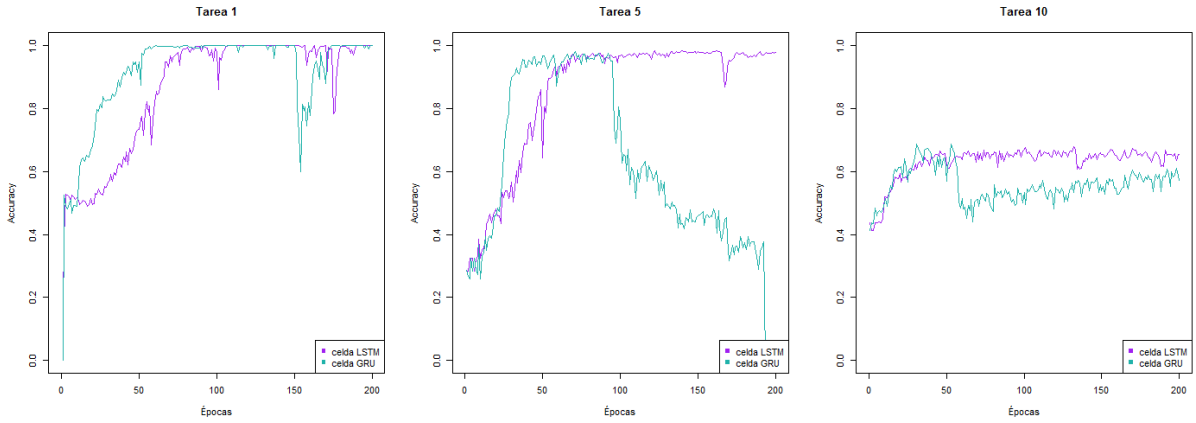


Figura 4.11: Resultados en función del tipo de celda.

Tras el análisis realizado en esta sección se puede concluir que los parámetros óptimos para la red neuronal recurrente, sobre este conjunto de datos son los que se muestran en la siguiente tabla:

Parámetros óptimos	
Número de épocas	200
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.01
Tamaño de inmersión	80
Tamaño de batch	32
Tipo de celda	LSTM

Tabla 4.7: Selección óptima de los parámetros.

Además, se presentan en la siguiente tabla los tiempos de procesamiento y entrenamiento para las tareas 1, 5 y 10, con la selección de parámetros óptima:

Comparaciones		
Tarea	**Tiempo procesamiento**	**Tiempo entrenamiento**
1	0.20 segs - 0.003 mins	480.46 segs - 8.01 mins
5	0.30 segs - 0.005 mins	4594.37 segs - 76.57 mins
10	0.21 segs - 0.003 mins	636.22 segs - 10.60 mins

Tabla 4.8: Tiempos de ejecución para el modelo *RNN*.

En la siguiente tabla aparecen los resultados tras evaluar el modelo en el conjunto test una vez ha sido entrenado. Estos resultados son los que se compararán con los que se obtengan para los otros modelos.

Red Neuronal Recurrente	
Tarea	**Accuracy en test**
1	0.993
2	0.3
3	0.241
4	0.724
5	0.326
6	0.497
7	0.488
8	0.336
9	0.638
10	0.674
11	0.92
12	0.917
13	0.935
14	0.342
15	0.213
16	0.449
17	0.52
18	0.469
19	0.125
20	0.977

Tabla 4.9: Resultados al evaluar el modelo *RNN* tras el entrenamiento.

Esta red consigue un acierto menor que 0.4 en 7 tareas, es decir en el 35 % de las situaciones planteadas, siendo el acierto medio del 55.4 %. Las tareas para las que se obtienen peores resultados son la 2, 3, 5, 8, 14, 15 y 19. Esto se debe a que de las redes consideradas, esta es la más sencilla. En la sección 4.3 se compararán estos resultados con los obtenidos por las otras redes.

4.2. Resultados MemN2N para tareas bAbI

Épocas

Si el número de épocas es mayor de lo necesario puede producirse sobreentrenamiento y en este caso el error empieza a aumentar. Se observa en todas las gráficas de la figura 4.12 que 100 épocas sería la elección óptima, dado que en todos los casos se obtienen mejores resultados que limitando el número de épocas a 60, y los mismos o incluso mejores que considerando 200 épocas. Además, puesto que considerar 200 épocas implica mayor tiempo de entrenamiento, se concluye que 100 son suficientes. Así pues, las posteriores comparaciones se realizarán con el número de épocas fijado a 100.

Parámetros del entrenamiento	
Número de épocas	60 - 100 - 200
Número de saltos	3
Inicialización de los pesos	normal
Algoritmo de optimización	SGD
Tasa de aprendizaje	0.01
Reducción de la tasa de aprendizaje	15
Norma del gradiente máxima	40
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.10: Parámetros seleccionados para el análisis del número de épocas.

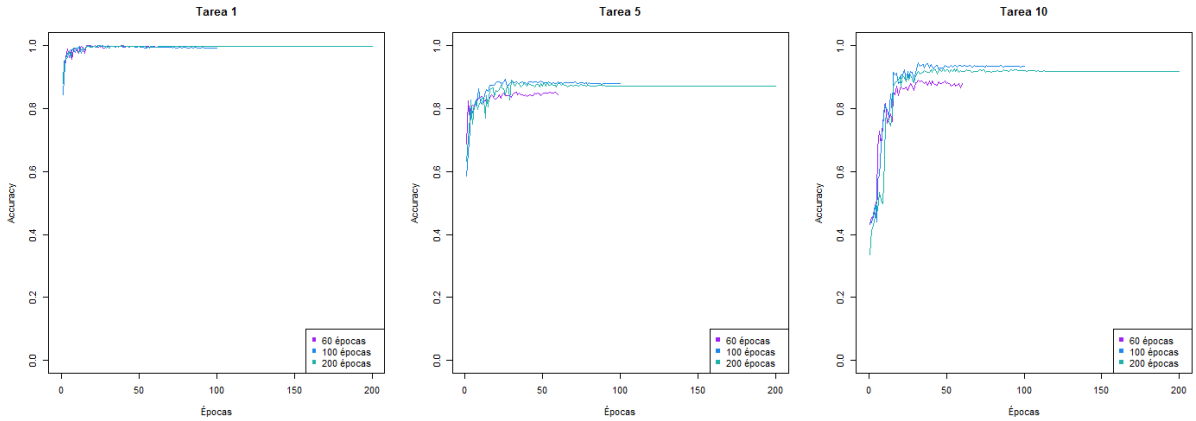


Figura 4.12: Resultados en función del número de épocas.

Saltos

El número de saltos es un parámetro que aparece en las redes de memoria y que no estaba presente en los análisis anteriores para la red neuronal recurrente. En este caso se muestran en la figura 4.13 los resultados para todas las tareas y ver claramente que un mejor ajuste del modelo depende del número de saltos.

Parámetros del entrenamiento	
Número de épocas	100
Número de saltos	1 - 3 - 8
Inicialización de los pesos	normal
Algoritmo de optimización	SGD
Tasa de aprendizaje	0.01
Reducción de la tasa de aprendizaje	15
Norma del gradiente máxima	40
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.11: Parámetros seleccionados para el análisis del número de saltos.

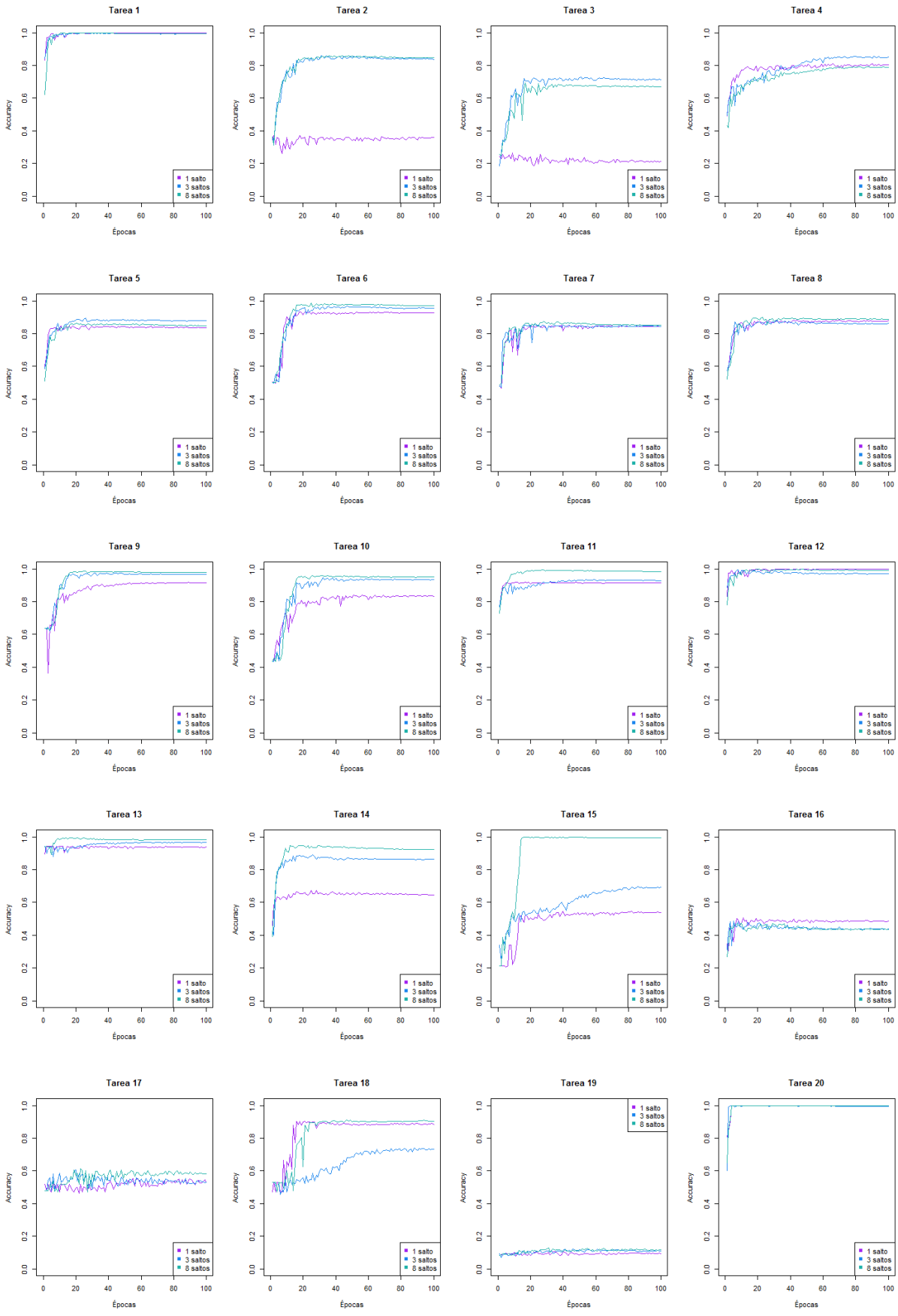


Figura 4.13: Resultados para cada tarea en función del número de saltos.

Se observa que a mayor número de saltos realizados, es decir cuantas más veces la red analiza todo el conjunto de datos, el acierto en las estimaciones de las respuestas es mayor.

Pero dado que el conjunto de datos tiene un tamaño considerable e interesa reducir los tiempos computacionales lo máximo posible. Los tiempos obtenidos para los tres saltos considerados se muestran en la tabla 4.12.

Comparaciones		
Saltos	**Tiempo procesamiento**	**Tiempo entrenamiento**
1	9.66 segs - 0.16 mins	2436.91 segs - 40.62 mins
3	10.22 segs - 0.17 mins	4006.35 segs - 66.77 mins
8	9.68 segs - 0.16 mins	6272.36 segs - 104.54 mins

Tabla 4.12: Comparación de tiempos de procesamiento y entrenamiento según el número de saltos.

Puesto que los resultados obtenidos para 3 y 8 saltos no eran muy dispares en la mayoría de los casos, pero si lo son los tiempos de entrenamiento, se considera el parámetro del número de saltos fijado a 3 para los posteriores análisis.

Vector de pesos inicial

Parámetros del entrenamiento	
Número de épocas	100
Número de saltos	3
Inicialización de los pesos	normal - Xavier uniforme - Xavier normal
Algoritmo de optimización	SGD
Tasa de aprendizaje	0.01
Reducción de la tasa de aprendizaje	15
Norma del gradiente máxima	40
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.13: Parámetros seleccionados para el análisis de la inicialización aleatoria de los pesos.

La red se ha entrenado a partir de tres inicializaciones aleatorias distintas de los pesos. La primera considerada es una inicialización aleatoria que sigue una distribución normal éstandar $N(0, 0.1)$, y las otras dos son las inicializaciones Xavier introducidas en la sección 3.2.1, en las que los pesos seguían una distribución uniforme y una normal.

Según los gráficos obtenidos para estos tres casos, mostrados en la figura 4.14, aunque las diferencias no son grandes se comprueba que da mejores resultados una inicialización aleatoria que siga una distribución normal, en lugar de uniforme. Para posteriores análisis se utilizará la inicialización aleatoria normal.

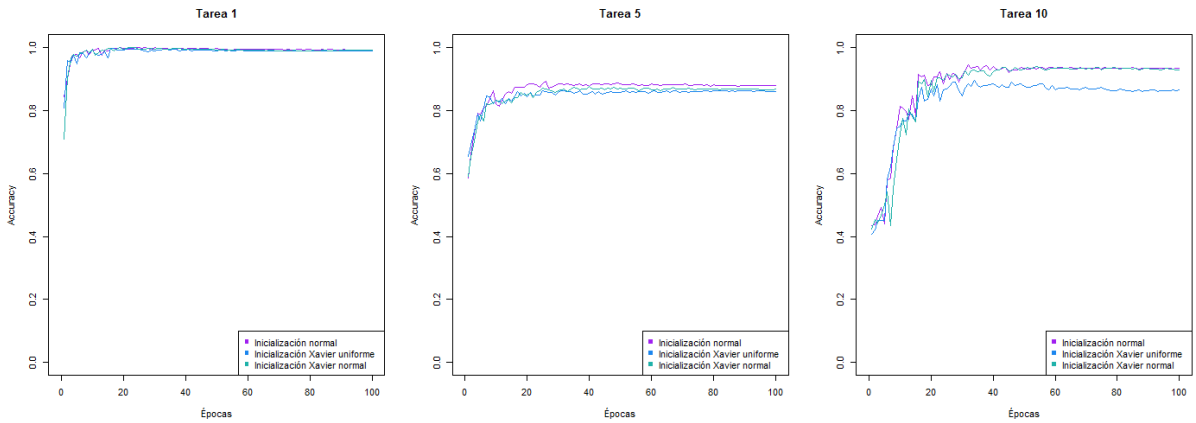


Figura 4.14: Resultados en función del número de épocas.

Tasa de aprendizaje

Parámetros del entrenamiento	
Número de épocas	100
Número de saltos	3
Inicialización de los pesos	normal
Algoritmo de optimización	SGD
Tasa de aprendizaje	0.1 - 0.01 - 0.001
Reducción de la tasa de aprendizaje	0 - 15 - 25
Norma del gradiente máxima	40
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.14: Parámetros seleccionados para el análisis de la tasa de aprendizaje y su reducción.

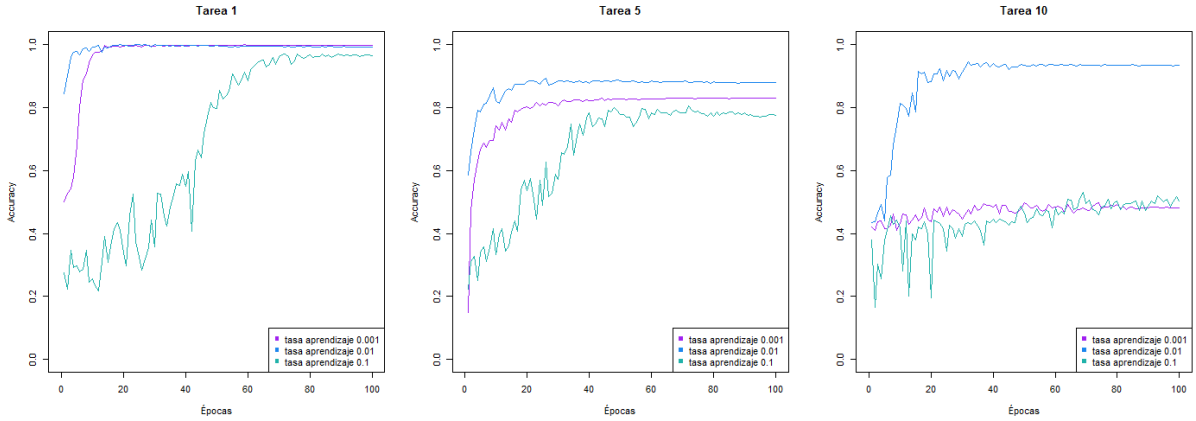


Figura 4.15: Resultados en función de la tasa de aprendizaje.

Se han comparado tres tasas de aprendizaje iniciales: 0.1, 0.01 y 0.001, consideradas como adaptativas; es decir cada cierto número de épocas, 15 en este caso, la tasa de aprendizaje se reduce a la mitad.

En la figura 4.15 puede verse que para una tasa de aprendizaje demasiado grande la curva del *accuracy* a lo largo de las épocas oscila bastante, debido a que influye en la adaptación de los pesos, haciendo que estos varíen bastante de una iteración a otra. También puede apreciarse que para algunas la curva ni siquiera alcanza un acierto del 50 % como en el caso de la tarea 10, luego no es una buena elección. Entre las otras dos se observa que tampoco es conveniente elegir una tasa demasiado pequeña, luego la elección más acertada para este conjunto de datos es 0.01 como tasa de aprendizaje inicial.

Si se considera la utilización de una tasa de aprendizaje fija frente a una tasa de aprendizaje adaptativa, como en la figura 4.16, se ve que la línea es más suave y alcanza valores superiores en el caso de una tasa que se va reduciendo conforme aumenta el número de épocas. Dado que se presentan diferencias, a continuación en la figura 4.17 se comparan resultados en función de cada cuántas épocas se reduce la tasa de aprendizaje a la mitad. Recordar que la reducción de la tasa de aprendizaje previene el sobreentrenamiento, como se dijo en 3.2.1.

Elegir que la reducción sea cada pocas épocas, por ejemplo 5, empeora bastante los resultados, pues en pocas épocas se reduce muchísimo y el aprendizaje pasa a ser demasiado lento. Esto sería similar a considerar una tasa de aprendizaje fija menor que 0.001, y ya se ha visto en la figura 4.15 que considerar una tasa de 0.001, y además con reducción, no daba buenos resultados. Por tanto, a la vista de los resultados mostrados en la figura 4.17, lo adecuado sería considerar que se redujese la tasa de aprendizaje al menos cada 15 épocas (con la elección de reducirla cada 25 épocas los resultados son parecidos pero observando con precisión se ve que el acierto es peor).

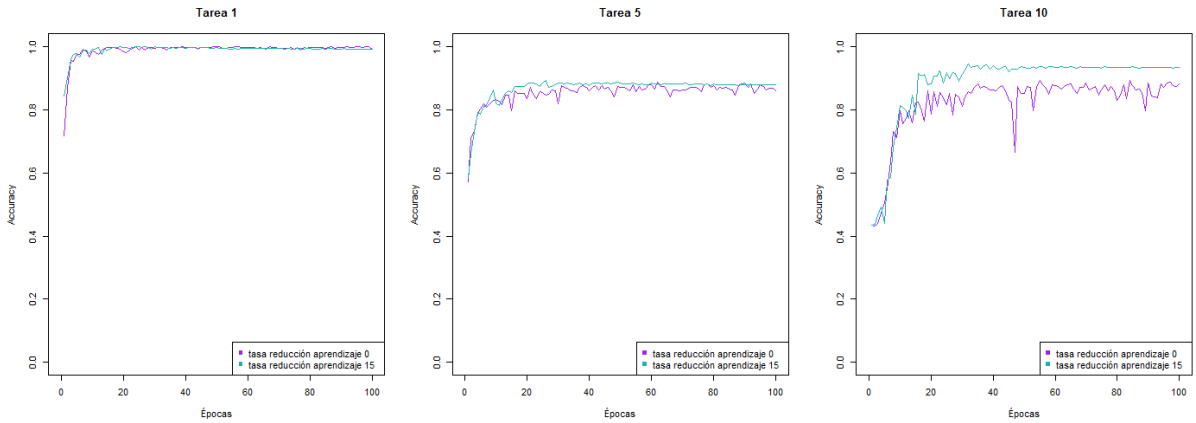


Figura 4.16: Resultados sin reducción de la tasa de aprendizaje.

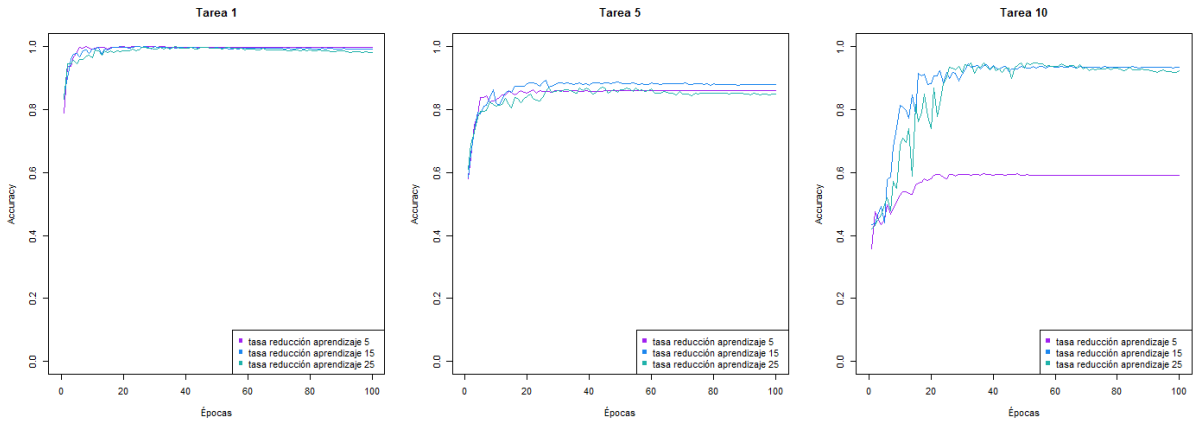


Figura 4.17: Resultados en función de la reducción de la tasa de aprendizaje.

Algoritmo de optimización

Parámetros del entrenamiento	
Número de épocas	100
Número de saltos	3
Inicialización de los pesos	normal
Algoritmo de optimización	SGD - Adam
Tasa de aprendizaje	0.01
Reducción de la tasa de aprendizaje	15
Norma del gradiente máxima	40
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.15: Parámetros seleccionados para el análisis del algoritmo de optimización.

Comparando los algoritmos de optimización del gradiente descendente estocástico y de la estimación del momento adaptativo (*Adam*), se obtiene que en este caso también es más adecuado considerar el primero de los algoritmos, aunque como puede verse en la figura 4.18 la diferencia no es significativa.

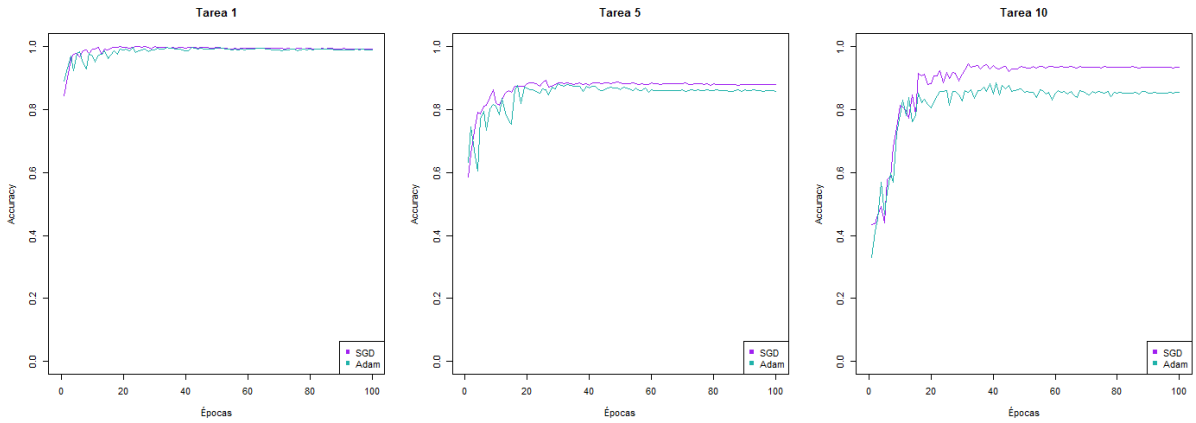


Figura 4.18: Resultados en función del algoritmo de optimización.

Tamaño de grupo o batch

Como se ha dicho en varias ocasiones, además de obtener buenos resultados, interesa que estos se alcancen en el menor tiempo posible. Uno de los parámetros que deben tenerse en cuenta a la hora de reducir los tiempos de entrenamiento es el tamaño del batch: cuanto más grande sea el grupo considerado, más rápido se analizarán los datos y se entrenará el modelo.

La figura 4.19 muestra que para un batch demasiado grande, en según qué situaciones el acierto no es bueno; es decir, resulta perjudicial considerar tanta información al mismo tiempo. Las pruebas realizadas con unos tamaños de batch 10 y 32 dan buenos porcentajes de acierto en ambos casos, siendo ligeramente mejor en el caso del tamaño 32. Además, por lo ya dicho, esta elección conlleva un menor tiempo de entrenamiento.

Parámetros del entrenamiento	
Número de épocas	100
Número de saltos	3
Inicialización de los pesos	normal
Algoritmo de optimización	SGD
Tasa de aprendizaje	0.01
Reducción de la tasa de aprendizaje	15
Norma del gradiente máxima	40
Tamaño de inmersión	50
Tamaño de batch	10 - 32 - 50
Tamaño de memoria	50

Tabla 4.16: Parámetros seleccionados para el análisis del tamaño del batch.

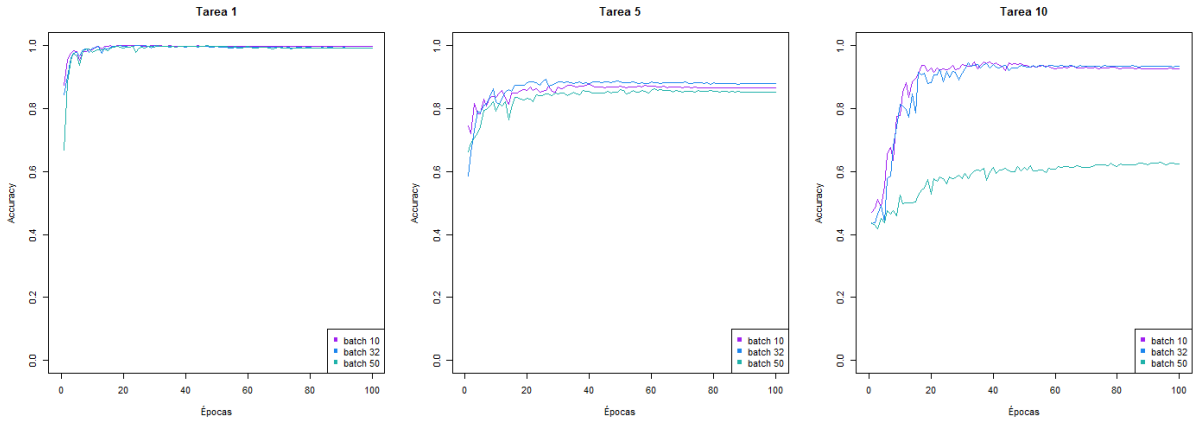


Figura 4.19: Resultados en función del tamaño de los grupos.

Norma del gradiente

Parámetros del entrenamiento	
Número de épocas	100
Número de saltos	3
Inicialización de los pesos	normal
Algoritmo de optimización	SGD
Tasa de aprendizaje	0.01
Reducción de la tasa de aprendizaje	15
Norma del gradiente máxima	20 - 40 - 80
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.17: Parámetros seleccionados para el análisis de la norma del gradiente máxima.

De acuerdo con lo mostrado en la figura 4.20, no es conveniente utilizar para normalizar el gradiente un valor muy pequeño. Dado que se obtienen resultados similares para una norma 40 y 80, incluso mejor para la elección de 40, este será el valor que se utilice para la norma del gradiente máxima para el resto de análisis.

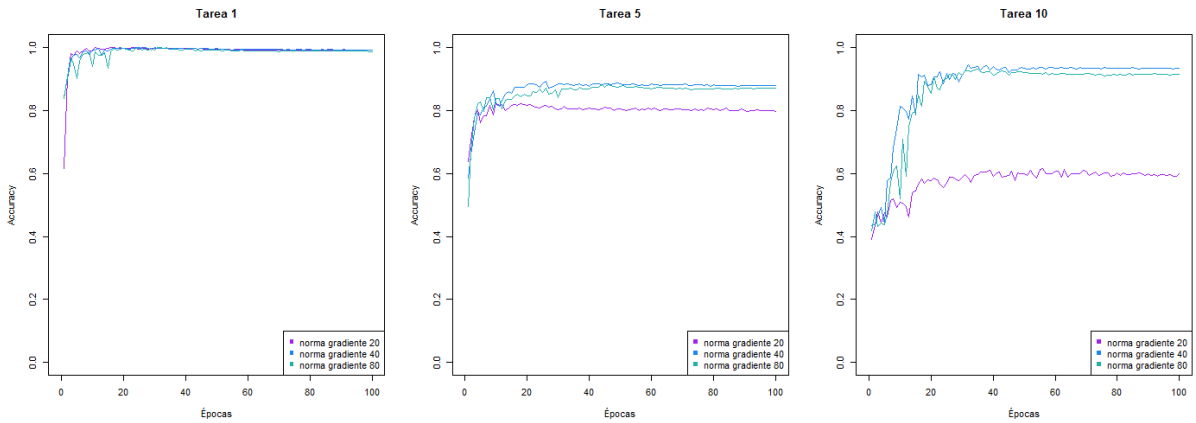


Figura 4.20: Resultados en función de la norma del gradiente.

Tamaño de inmersión o embedding

Parámetros del entrenamiento	
Número de épocas	100
Número de saltos	3
Inicialización de los pesos	normal
Algoritmo de optimización	SGD
Tasa de aprendizaje	0.01
Reducción de la tasa de aprendizaje	15
Norma del gradiente máxima	40
Tamaño de inmersión	20 - 40 - 80
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.18: Parámetros seleccionados para el análisis del tamaño de inmersión.

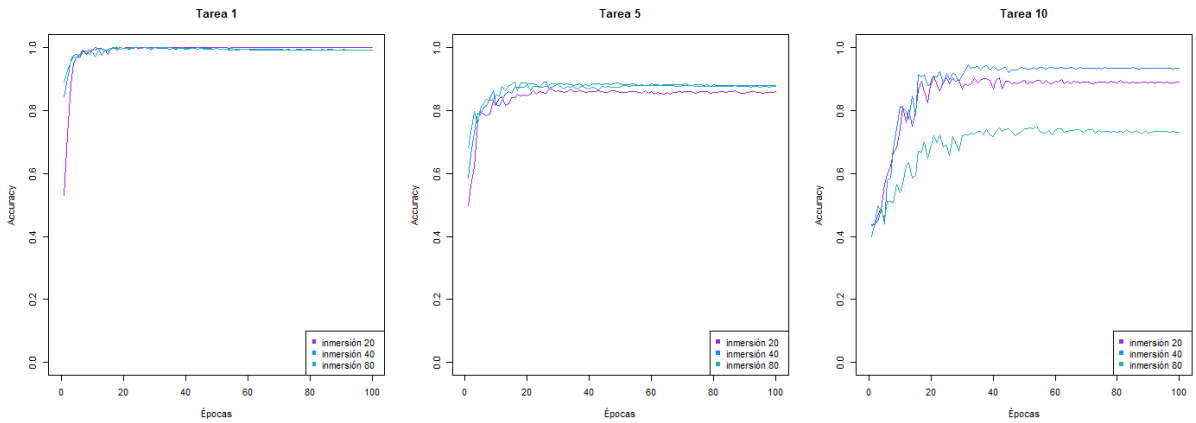


Figura 4.21: Resultados en función del tamaño de inmersión.

Para la elección del tamaño de inmersión, se consideran tres opciones: tamaño 20, 40 y 80. En la figura 4.21 se observa que en este caso un tamaño de inmersión 40 permite obtener buenos resultados en todos los casos. No es necesario considerar la dimensión del espacio de inmersión tan grande como pasaba para el modelo anterior, ya que el modelo actual cuenta con una nueva componente que el

anterior no tenía, la memoria. Al poder almacenar la información en memorias no se tiene la necesidad de considerar tantas neuronas en las capas ocultas como ocurría antes.

Tamaño de memoria

Parámetros del entrenamiento	
Número de épocas	100
Número de saltos	3
Inicialización de los pesos	normal
Algoritmo de optimización	SGD
Tasa de aprendizaje	0.01
Reducción de la tasa de aprendizaje	15
Norma del gradiente máxima	40
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	20 - 50 - 80

Tabla 4.19: Parámetros seleccionados para el análisis del tamaño de memoria.

Como se explicó en la introducción de los parámetros presenten en las redes en 3.2.1, el tamaño de la memoria mide el número de frases más recientes con información que son almacenadas. Esto resulta útil ya que los conjuntos de datos con los que se trabaja incluyen relaciones de información entre unas frases y otras previas. Se ha probado a considerar memorias de tamaño 20, 50 y 80, y en 4.22 se puede ver que la elección más adecuada es que las memorias tengan un tamaño intermedio de 50, es decir que puedan almacenarse las últimas 50 frases analizadas.

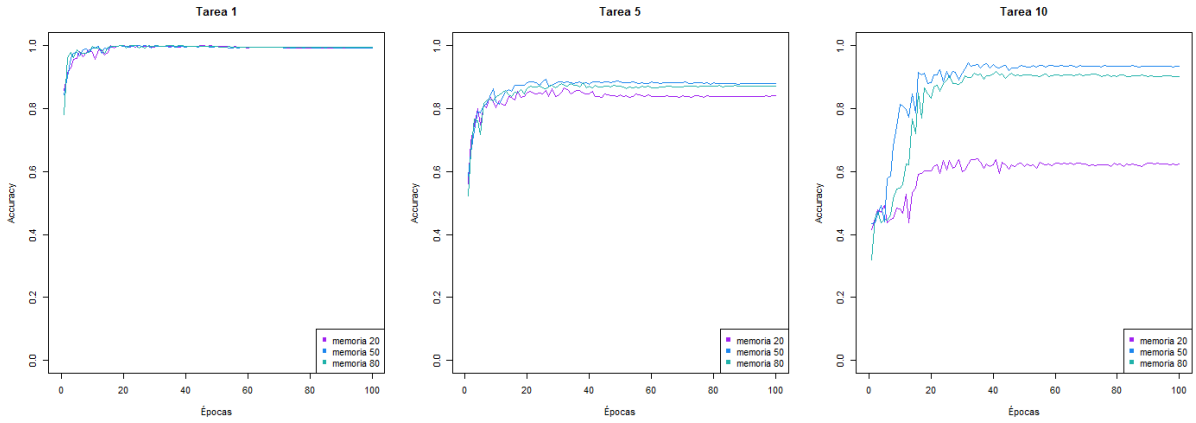


Figura 4.22: Resultados en función del tamaño de la memoria.

Después de los análisis realizados en esta sección se concluye que los parámetros óptimos para la red de memoria *end-to-end*, sobre este conjunto de datos son los que se muestran en la tabla 4.20. A continuación de esto se muestran en la tabla 4.21 los tiempos de procesamiento y entrenamiento para la selección de las tareas 1, 5 y 10 y para las 20 tareas conjuntamente, de acuerdo con la selección de parámetros óptima. Los tiempos de procesamiento son similares a los obtenidos para la *RNN*, sin embargo los tiempos de entrenamiento se reducen considerablemente, sobre todo para las tareas 5 y 10.

La tabla 4.22 presenta los resultados tras evaluar la red una vez ha sido entrenada. Estos resultados son los que se compararán con los que se obtengan para los otros modelos.

Parámetros óptimos	
Número de épocas	100
Número de saltos	3
Inicialización de los pesos	normal
Algoritmo de optimización	SGD
Tasa de aprendizaje	0.01
Reducción de la tasa de aprendizaje	15
Norma del gradiente máxima	40
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.20: Selección óptima de los parámetros.

Comparaciones		
Tarea	**Tiempo procesamiento**	**Tiempo entrenamiento**
1	0.18 segs - 0.003 mins	75.83 segs - 1.26 mins
5	0.87 segs - 0.01 mins	88.46 segs - 1.47 mins
10	0.28 segs - 0.005 mins	49.54 segs - 0.83 mins
Todas	7.38 segs - 0.12 mins	2140.82 segs - 35.68 mins

Tabla 4.21: Tiempos de ejecución para el modelo *MemN2N*.

Red de Memoria End-to-End	
Tarea	**Accuracy en test**
1	0.983
2	0.86
3	0.714
4	0.798
5	0.857
6	0.937
7	0.825
8	0.903
9	0.938
10	0.907
11	0.888
12	0.97
13	0.913
14	0.933
15	0.477
16	0.468
17	0.57
18	0.85
19	0.114
20	0.998

Tabla 4.22: Resultados al evaluar el modelo *MemN2N* tras el entrenamiento.

En este caso, la red únicamente alcanza un *accuracy* inferior a 0.4 en una única tarea, por tanto mejora con creces a la red considerada anteriormente. Esto es debido a que la *MemN2N* cuenta con la componente de memoria a la que se puede ir accediendo mediante los saltos, siendo esto conveniente por ejemplo en el caso de las tareas que precisaban de dos o tres argumentos para responder a una pregunta. En más de la mitad de las tareas el acierto obtenido está por encima del 85 %, y el acierto medio de la red para este conjunto de datos es del 79.5 %.

4.3. Resultados KV-MemNN para tareas bAbI

Épocas

Parámetros del entrenamiento	
Número de épocas	60 - 100 - 200
Número de saltos	3
Inicialización de los pesos	Xavier uniforme
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.01 - 0.001
Norma del gradiente máxima	40
Tamaño de características	40
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.23: Parámetros seleccionados para el análisis del número de épocas .

Para la selección del número de épocas, se estudiarán los resultados en función de la tasa de aprendizaje, considerando esta como 0.01 y 0,001, como puede verse en las figuras 4.23 y 4.24, respectivamente, ya que al probar inicialmente con la tasa 0,01 se vio que en algunos casos era insuficiente ya que el aprendizaje se quedaba estancado, como puede verse para la tarea 10.

En ambos casos, pero de manera más clara en la figura 4.24, se observa que tanto 60 como 100 épocas son escasas, ya que en alguna de las tareas, hasta la época 200 se siguen mejorando los resultados. Por tanto para abarcar el máximo aprendizaje posible en todos los casos, se considerará 200 como el número de épocas óptimo.

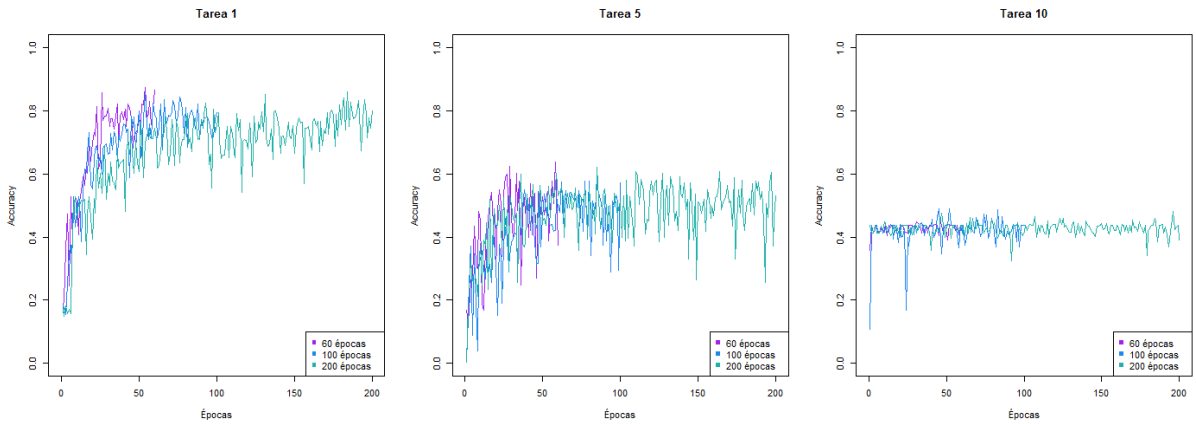


Figura 4.23: Resultados en función del número de épocas, con tasa de aprendizaje 0,01.

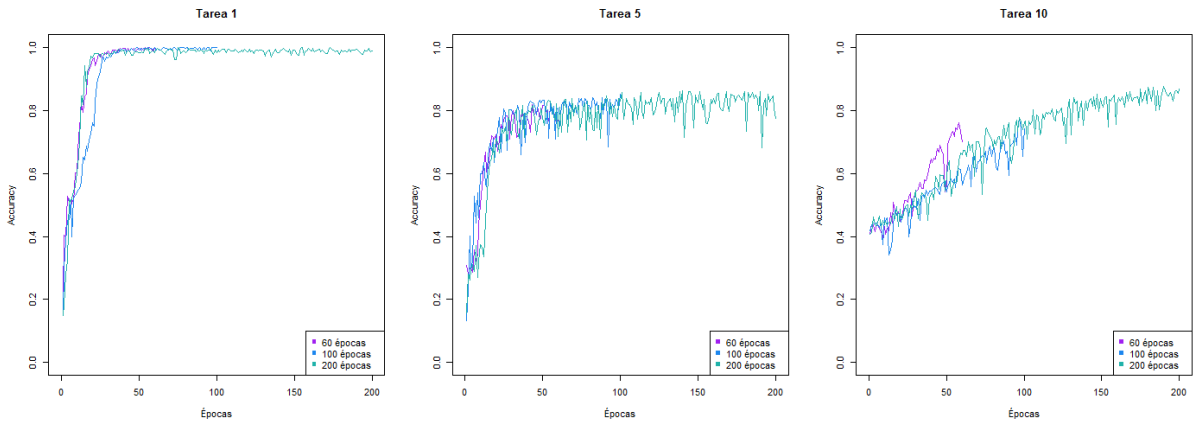


Figura 4.24: Resultados en función del número de épocas, con tasa de aprendizaje 0,001.

Saltos

Parámetros del entrenamiento	
Número de épocas	500
Número de saltos	1 - 3 - 8
Inicialización de los pesos	Xavier uniforme
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.001
Norma del gradiente máxima	40
Tamaño de características	50
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.24: Parámetros seleccionados para el análisis del número de saltos .

De igual manera que pasaba con los otros dos modelos, el número de saltos necesario que se establece para que la red analice todo el conjunto de datos basta con fijarlo a 3. Además de que en la figura 4.25 se ve que los resultados son mejores para todas las tareas cuando se consideran 3 saltos, esta elección es preferible frente a la de 8 saltos ya que supone un menor tiempo de entrenamiento.

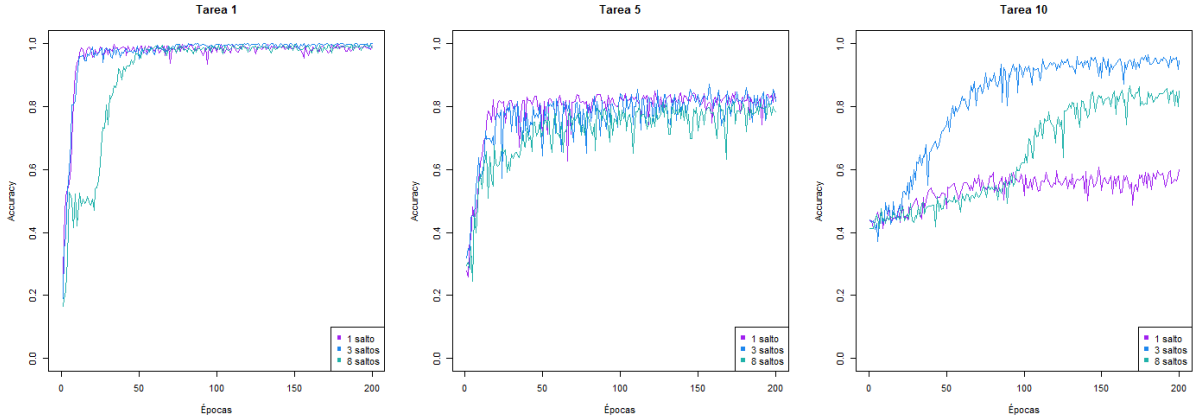


Figura 4.25: Resultados en función del número de saltos.

Vector de pesos inicial

Parámetros del entrenamiento	
Número de épocas	200
Número de saltos	3
Inicialización de los pesos	normal - Xavier uniforme - Xavier normal
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.001
Norma del gradiente máxima	20
Tamaño de características	50
Tamaño de inmersión	40
Tamaño de batch	50
Tamaño de memoria	50

Tabla 4.25: Parámetros seleccionados para el análisis del método de inicialización aleatoria de los pesos.

Para este modelo también es conveniente que la inicialización aleatoria de los pesos siga una distribución normal, pero en este caso la distribución normal de Xavier, no la normal estándar como se había seleccionado en el modelo anterior. La figura 4.26 muestra que en cualquier caso son preferibles las inicializaciones Xavier (introducidas en la sección 3.2.1) a la normal estándar, y entre ellas proporciona mejores resultados la Xavier normal.

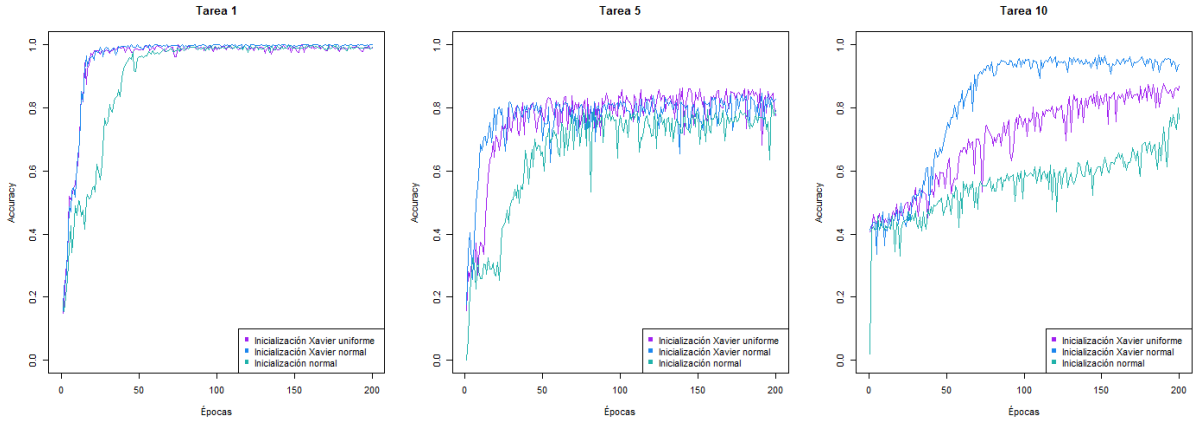


Figura 4.26: Resultados en función de la inicialización de los pesos.

Tasa de aprendizaje

Se comparan a continuación en la figura 4.27 las tasas de aprendizaje 0.1, 0.01 y 0.001. Aquí se muestra que un aprendizaje demasiado rápido (el correspondiente a la tasa 0.1) no es para nada adecuado, ya que el acierto oscila mucho pero siempre entorno a un valor medio casi 0. Conviene en este caso elegir la tasa de aprendizaje más baja, es decir 0.001, pues es la que permite que la red siga aprendiendo a lo largo de las épocas, cosa que no pasa con la 0.01 que hace que en algunas tareas se estanque el aprendizaje. Así pues 0.001 será la tasa de aprendizaje elegida como óptima para que la red aprenda sobre este conjunto de datos.

Parámetros del entrenamiento	
Número de épocas	200
Número de saltos	3
Inicialización de los pesos	Xavier uniforme
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.1 - 0.01 - 0.001
Norma del gradiente máxima	20
Tamaño de características	50
Tamaño de inmersión	40
Tamaño de batch	50
Tamaño de memoria	50

Tabla 4.26: Parámetros seleccionados para el análisis de la tasa de aprendizaje.

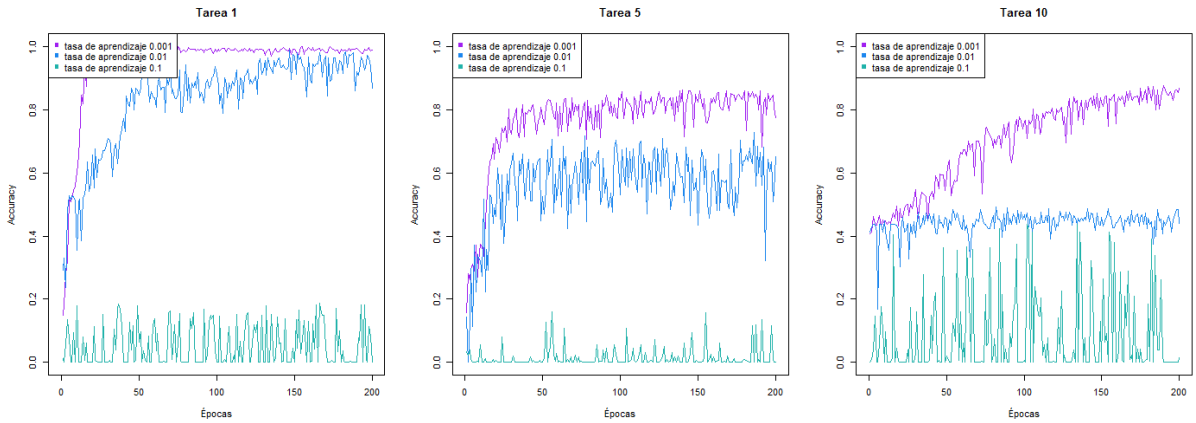


Figura 4.27: Resultados en función de la tasa de aprendizaje.

Algoritmo de optimización

Parámetros del entrenamiento	
Número de épocas	200
Número de saltos	3
Inicialización de los pesos	Xavier uniforme
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.001
Norma del gradiente máxima	20
Tamaño de características	50
Tamaño de inmersión	40
Tamaño de batch	50
Tamaño de memoria	50

Tabla 4.27: Parámetros seleccionados para el análisis del algoritmo de optimización.

Se comparan ahora en la figura 4.28 los dos algoritmos de optimización. En todos los casos resulta más conveniente una actualización de los pesos mediante el algoritmo Adam. También se comparan en la figura 4.29 distintos valores para el parámetro epsilon del algoritmo Adam. En la sección 3.2.1 se explicó que este parámetro evitaba la división por cero del factor que acompaña a la tasa de aprendizaje

en la actualización del peso. Se ve que interesa que este valor sea lo más pequeño posible, pues así la reducción de este factor es mínima y tendrá más influencia en la actualización.

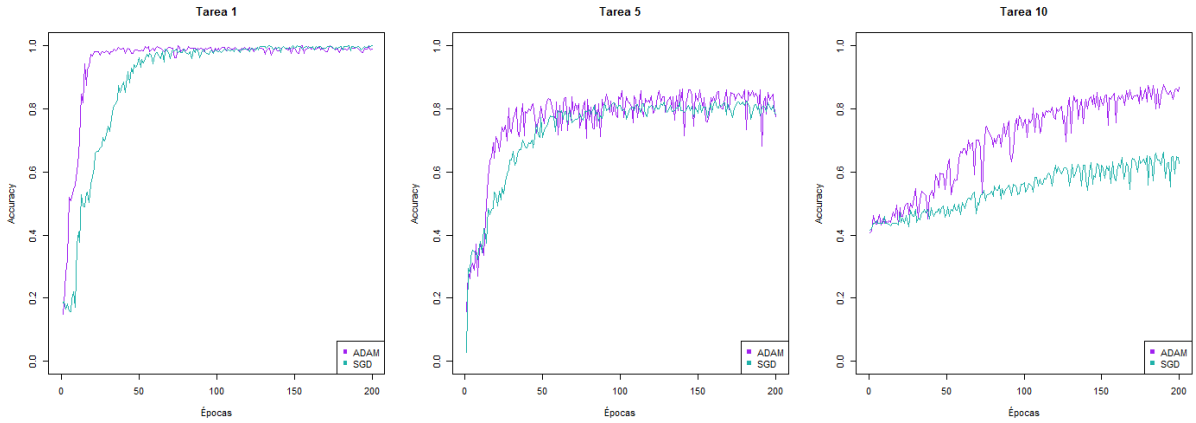


Figura 4.28: Resultados en función del método de optimización.

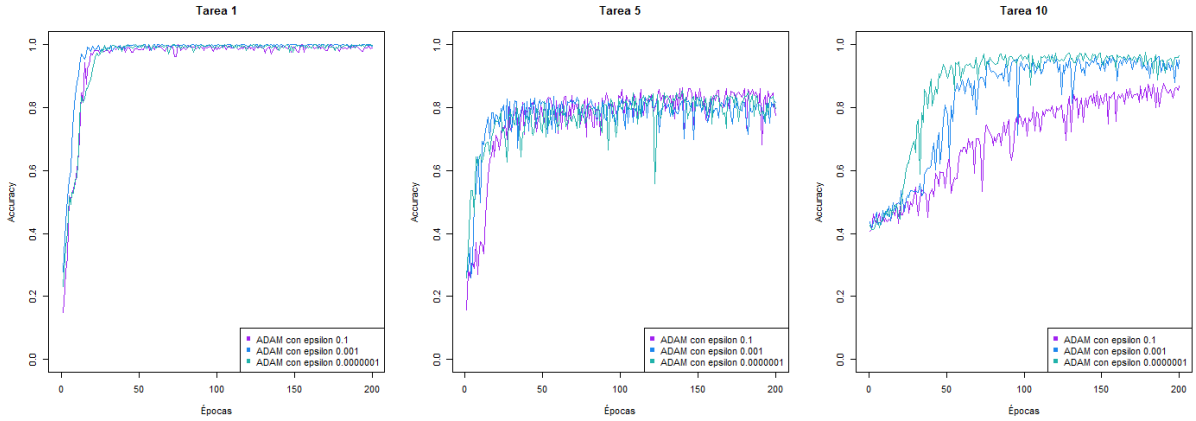


Figura 4.29: Resultados en función del valor de epsilon para el método Adam.

Tamaño de grupo o batch

Parámetros del entrenamiento	
Número de épocas	200
Número de saltos	3
Inicialización de los pesos	Xavier uniforme
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.001
Norma del gradiente máxima	40
Tamaño de características	50
Tamaño de inmersión	40
Tamaño de batch	10 - 32 - 50
Tamaño de memoria	50

Tabla 4.28: Parámetros seleccionados para el análisis del tamaño del batch.

Por otro lado, como se ha visto en los análisis de los otros modelos y como se ve en la figura 4.30, el tamaño del batch si que es un parámetro que influye notablemente a la hora de obtener mejores o peores resultados. Se verá más adelante que de la elección óptima de este parámetro puede depender la de otros y viceversa. En este caso conviene elegir un tamaño de batch por encima de 10, ya que este valor hace que el aprendizaje se estanque. A la vista de los resultados, podría elegirse tanto el batch de tamaño 32 como el de tamaño 50. La elección final de este parámetro puede depender de la elección de otros parámetros, y si no, del tiempo de entrenamiento. Entrenar el modelo con un batch de tamaño 50 será más rápido que con uno de tamaño 32, como puede verse en la tabla 4.29.

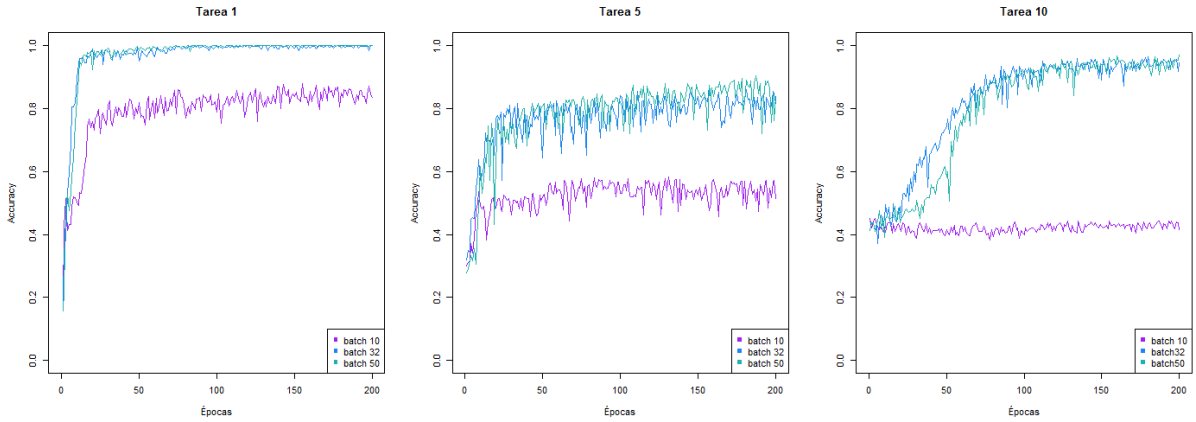


Figura 4.30: Resultados en función del tamaño del batch.

Comparaciones		
Batch	**Tiempo procesamiento**	**Tiempo entrenamiento**
32	6.49 segs - 0.11 mins	4214.40 segs - 70.24 mins
50	7.98 segs - 0.13 mins	3777.16 segs - 62.95 mins

Tabla 4.29: Comparación de tiempos de procesamiento y entrenamiento según el tamaño del batch.

Norma del gradiente

Parámetros del entrenamiento	
Número de épocas	200
Número de saltos	3
Inicialización de los pesos	Xavier uniforme
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.001
Norma del gradiente máxima	20 - 40 - 80
Tamaño de características	50
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.30: Parámetros seleccionados para el análisis de la norma del gradiente máxima.

En cuanto a la norma del gradiente, en la figura 4.31 se muestra que no es un parámetro que afecte demasiado a los resultados, y que la elección óptima sería que la norma máxima fuese 40.

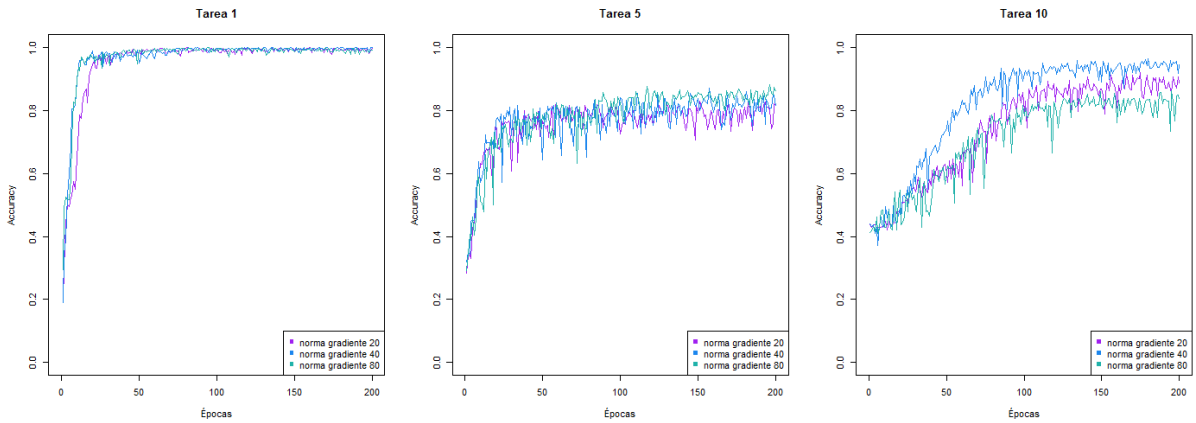


Figura 4.31: Resultados en función de la norma del gradiente.

Tamaño de características o features

Parámetros del entrenamiento	
Número de épocas	200
Número de saltos	3
Inicialización de los pesos	Xavier uniforme
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.001
Norma del gradiente máxima	40
Tamaño de características	10 - 20 - 50
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.31: Parámetros seleccionados para el análisis del tamaño de las características.

Este parámetro representa el tamaño del vector al que se codifica la información de entrada, antes de transformarlo al espacio de inmersión correspondiente. Por tanto, como el tamaño de las características y el tamaño de inmersión influyen en la forma de representación y transformación de los datos, se verá en el análisis siguiente que van a ser parámetros dependientes.

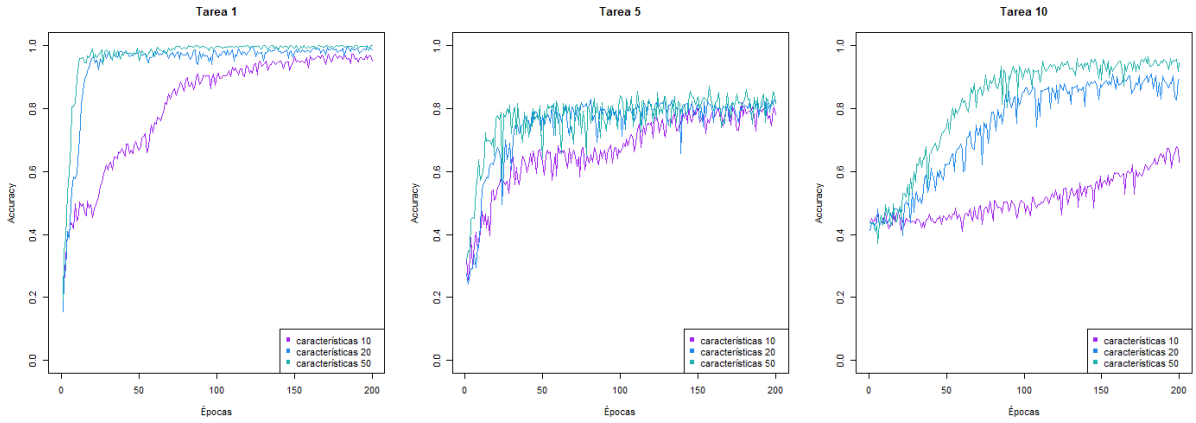


Figura 4.32: Resultados en función del tamaño de las características.

Previo a esto, si se analiza por separado la influencia del tamaño de las características, se ve en la figura 4.32 que en este caso los tamaños 10 y 20 son insuficientes para que las representaciones sean precisas y el modelo capte toda la información.

Tamaño de inmersión o embedding

Parámetros del entrenamiento	
Número de épocas	200
Número de saltos	3
Inicialización de los pesos	Xavier uniforme
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.001
Norma del gradiente máxima	40
Tamaño de características	50
Tamaño de inmersión	20 - 40 - 80
Tamaño de batch	32
Tamaño de memoria	50

Tabla 4.32: Parámetros seleccionados para el análisis del tamaño de inmersión.

De la misma manera que con el tamaño de las características, se comparan individualmente en la figura 4.33 diferentes elecciones para el tamaño de inmersión. Según los parámetros fijados en la tabla 4.32 el valor óptimo de este tamaño sería 40 o incluso 80 (para la tarea 5 este último tamaño proporciona un acierto mayor, y para las otras dos los resultados son prácticamente iguales). Como se ha dicho antes, el tamaño de las características y el tamaño de inmersión están relacionados, luego en la figura 4.34 se hace una comparación conjunta considerando los dos mejores tamaños de características que se habían seleccionado y los tres tamaños de inmersión que se acaban de analizar. A la vista de los resultados aquí mostrados, para la elección de tamaño de característica 20, convendría elegir un tamaño de inmersión 80, y para la elección de tamaño de característica 50, convendría elegir un tamaño de inmersión 40 u 80. De estas comparaciones conjuntas se concluye que los parámetros óptimos serían tamaño de características 50 y tamaño de inmersión 40.

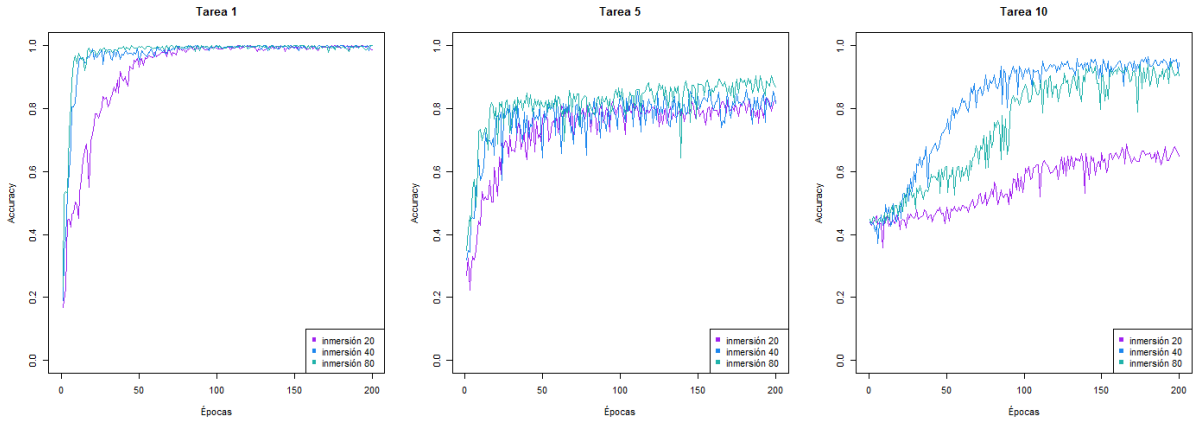


Figura 4.33: Resultados en función del tamaño de la inmersión.

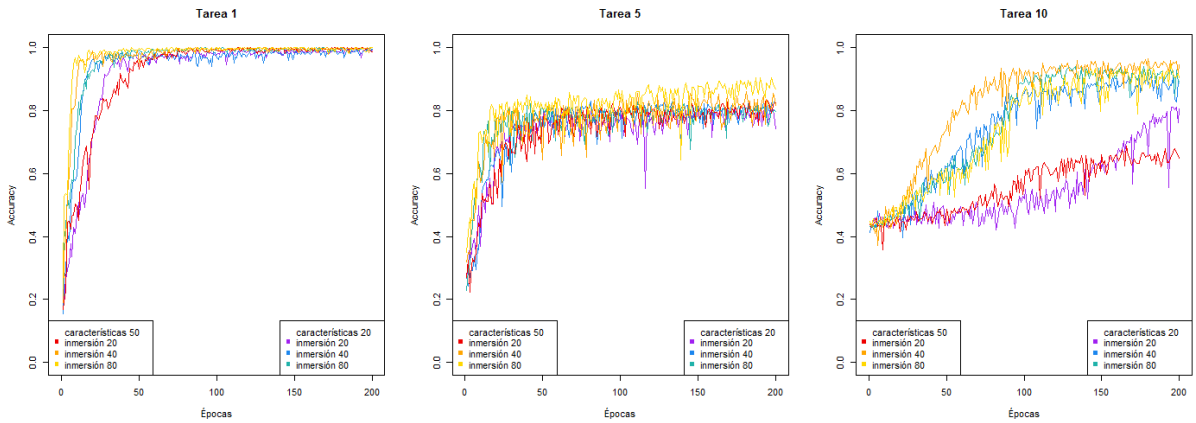


Figura 4.34: Resultados en función del tamaño de las características y la inmersión.

Tamaño de memoria

Parámetros del entrenamiento	
Número de épocas	200
Número de saltos	3
Inicialización de los pesos	Xavier uniforme
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.001
Norma del gradiente máxima	40
Tamaño de características	50
Tamaño de inmersión	40
Tamaño de batch	32
Tamaño de memoria	20 - 50 - 80

Tabla 4.33: Parámetros seleccionados para el análisis del tamaño de memoria.

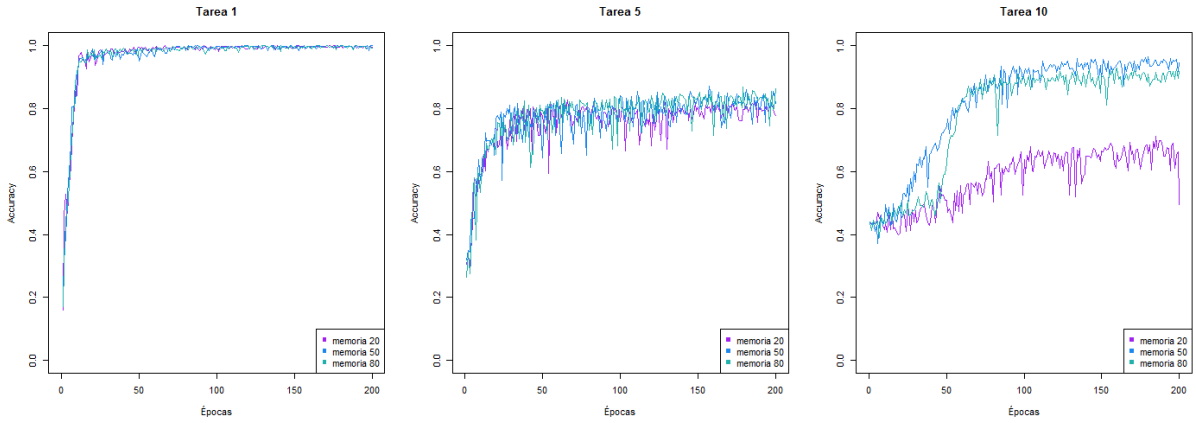


Figura 4.35: Resultados en función del tamaño de la memoria.

En cuanto al tamaño de la memoria, como se hizo para la red *MemN2N*, se han considerado las opciones 20, 50 y 80. En la figura 4.35 se ve aprecia que el tamaño de memoria 20 no va a ser suficiente en según que casos, y dado que considerando 50 u 80 el acierto del modelo es bastante parecido, se van

a comparar los tamaños también en función de la elección del batch. El análisis conjunto de estos parámetros se muestra en la figura 4.36. Los resultados obtenidos son muy similares, pero puede apreciarse que considerar un batch de tamaño 50 junto con una memoria de tamaño 50 hace que el modelo alcance un acierto mayor.

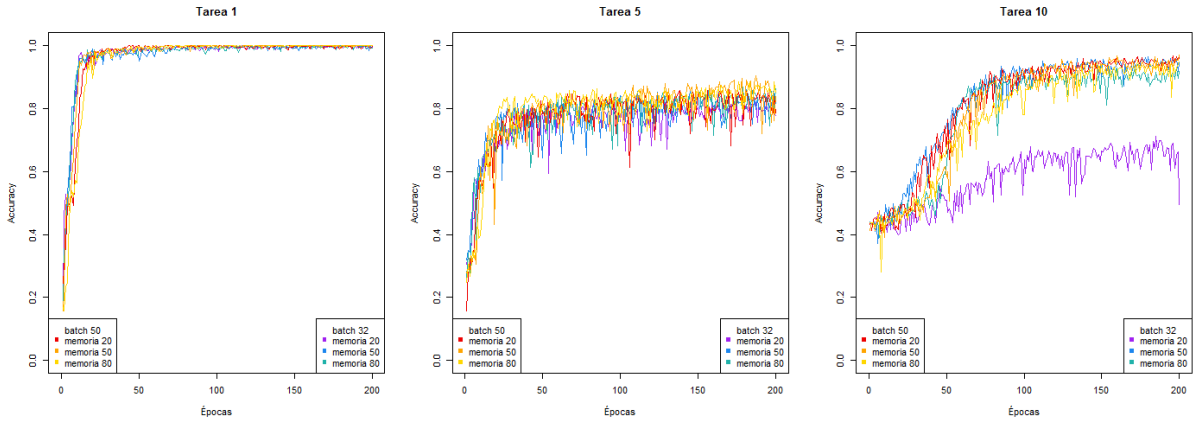


Figura 4.36: Resultados en función del tamaño de la memoria y del batch.

Por tanto, los parámetros óptimos para la red de memoria clave-valor, sobre este conjunto de datos son:

Parámetros óptimos	
Número de épocas	200
Número de saltos	3
Inicialización de los pesos	Xavier normal
Algoritmo de optimización	Adam
Tasa de aprendizaje	0.001
Norma del gradiente máxima	40
Tamaño de características	50
Tamaño de inmersión	40
Tamaño de batch	50
Tamaño de memoria	50

Tabla 4.34: Selección óptima de parámetros para el modelo *KV-MemNN*.

Se muestran también en la siguiente tabla los tiempos de procesamiento y entrenamiento para la selección de las tareas 1, 5 y 10 y para las 20 tareas conjuntamente, considerando la selección de parámetros óptima.

Comparaciones		
Tarea	**Tiempo procesamiento**	**Tiempo entrenamiento**
1	0.18 segs - 0.003 mins	54.25 segs - 0.90 mins
5	0.46 segs - 0.008 mins	124.68 segs - 2.08 mins
10	0.21 segs - 0.003 mins	59.17 segs - 0.99 mins
Todas	9.07 segs - 0.15 mins	3634.20 segs - 60.57 mins

Tabla 4.35: Tiempos de ejecución para el modelo *KV-MemNN*.

Ambos tiempos son bastante similares a los mostrados para el modelo *MemN2N*.

En la tabla a continuación aparecen los resultados tras evaluar la red una vez ha sido entrenada. Estos resultados se compararán con los que obtenidos para los otros modelos.

Red de Memoria Clave-Valor	
Tarea	**Accuracy en test**
1	0.998
2	0.793
3	0.569
4	0.942
5	0.837
6	0.969
7	0.844
8	0.902
9	0.973
10	0.968
11	0.968
12	0.992
13	0.985
14	0.902
15	0.486
16	0.471
17	0.547
18	0.772
19	0.076
20	0.999

Tabla 4.36: Resultados al evaluar el modelo *KV-MemNN* tras el entrenamiento.

Esta red obtiene un *accuracy* superior a 0.4 en todas las tareas excepto una, la tarea 19 igual que ocurría con la red anterior. En este caso también se obtiene un acierto superior al 85 % en más de la mitad de las tareas, sin embargo el acierto medio asciende al 79.9 %.

Comparación de resultados

Los resultados obtenidos para cada una de las tareas al validar los modelos se muestran conjuntamente en la tabla 4.37.

En la mayoría de las tareas, sobre todo en las que presentan mayor dificultad ya que reflejan correlaciones o dependencias entre las diferentes frases que forman las historias correspondientes a cada pregunta (tareas 2, 3, 5, 8, 15), los modelos de redes de memoria superan con creces a la red neuronal recurrente, tanto en acierto como en tiempo de entrenamiento. El porcentaje de acierto de las redes de memoria ascendía un 24 % con respecto de la red neuronal recurrente.

Comparando ahora ambas redes de memoria, los resultados son bastante similares, pero cabe decir que la clave-valor supera a la *end-to-end* en número de tareas y en porcentaje medio de acierto, aunque bien puede observarse en la tabla 4.37 que en determinadas tareas, como son la 2, 3, 14, 18 y 19, la segunda red sobrepasa a la primera.

Comparaciones accuracy			
Tarea	**RNN**	**MemN2N**	**KVMemNN**
1	0.993	0.983	0.998
2	0.3	0.86	0.793
3	0.241	0.714	0.569
4	0.724	0.798	0.942
5	0.326	0.857	0.837
6	0.497	0.937	0.969
7	0.488	0.825	0.844
8	0.336	0.903	0.902
9	0.638	0.938	0.973
10	0.674	0.907	0.968
11	0.92	0.888	0.968
12	0.917	0.97	0.992
13	0.935	0.913	0.985
14	0.342	0.933	0.902
15	0.213	0.477	0.486
16	0.449	0.468	0.471
17	0.52	0.57	0.547
18	0.469	0.85	0.772
19	0.125	0.114	0.076
20	0.977	0.998	0.999

Tabla 4.37: Comparaciones de los tres modelos para cada una de las tareas.

Los parámetros más influyentes en las tres redes son, sobre todo, la tasa de aprendizaje y el número de épocas, y en el caso de las redes de memoria influye en los resultados también el número de saltos. Luego, otros parámetros que contribuyen a la mejora de los resultados son el tamaño del batch y en el caso de las redes de memoria también el tamaño de inmersión y el tamaño de memoria, pero la elección de estos parámetros está supedidata al tamaño muestral. Por ejemplo, los tamaños de batch seleccionados para este conjunto de datos son 32 y 50 dado que el tamaño del conjunto de datos aquí considerado es mucho más grande que el considerado en la siguiente sección, donde ha sido suficiente elegir un batch de tamaño 10.

4.4. Resultados MemN2N y KV-MemNN para WikiMovies

Esta sección va a centrarse en el análisis de las redes de memoria estudiadas en el conjunto de datos *WikiMovies*, ya que son las más adecuadas para tratar datos textuales, como se ha visto en los análisis previos. Más concretamente, se van a comparar las diferentes representaciones de la información introducidas en 2.4.2: el formato frase, donde los datos son las diferentes oraciones que puedan formar un texto; el formato ventana, que sería el resultante al separar el texto completo en conjuntos de W palabras, seleccionando a partir de una palabra varias que le suceden y preceden; y el formato tripleta, que es como puede encontrarse la información en las bases de conocimiento, con la estructura de "*sujeto-predicado-objeto*". Además, se considera la opción de tener parte de información en forma de frases y que se añada una información extra en formato tripleta, para así comprobar que hacer confluir estos dos formatos permite obtener mejores resultados. La motivación última es que pueda completarse una determinada información de texto completo con nueva información obtenida a posteriori. Esta información adicional se incluye en forma de tripletas, ya que este formato requiere menos capacidad de almacenaje, y se procesa y entrena más rápido.

Todos los análisis que van a presentarse se han realizado con la selección óptima de parámetros para cada modelo. No se presentan en la memoria los resultados que han llevado a optar por esas selecciones, pero se presentan en las tablas 4.38 y 4.39. El parámetro que destaca sobremanera es el tamaño de la memoria, ya que depende del número máximo de elementos que constituyen la información de cada película: se necesitan más celdas de memoria que almacenen la información más reciente que pueda ser de utilidad para responder a una pregunta.

Se ha considerado un subconjunto de datos para hacer las diferentes comparaciones en unos tiempos razonables, suficiente para apreciar las diferencias. Este conjunto contiene información sobre 38 películas. En cuanto a las preguntas y respuestas, para el entrenamiento se tienen 522, para la validación 70 y para la evaluación o testeo 327. El fichero que contiene la información en formato frase está formado por 312 elementos, el de formato ventana 2267 y el de formato tripleta 321.

Dado que para responder a determinadas preguntas se necesita información de hasta tres películas distintas, se han considerado tamaños de memoria que sean el doble o triple del número máximo de elementos con información sobre una película, o en su defecto el número medio, dependiendo del formato en que se presente dicha información. En el caso del formato frase la mejor opción es considerar el triple del tamaño máximo del número de elementos presentes, ya que es preciso contar con más número de datos para poder obtener información más acertada. Este número máximo es 25, luego el tamaño de memoria óptimo será 75; es decir se almacenarán hasta 75 elementos (frases). También lo será cuando se considere el formato conjunto de frase y tripleta, al estar considerándose este primer formato.

Por otro lado, para los formatos de ventana y tripleta, se considera el número medio de elementos en lugar del máximo, 26 y 7 respectivamente. Para almacenar la información de tipo ventana bastará con considerar el doble, es decir las memorias serán de tamaño 52. Para las tripletas se considerará el triple del valor medio, luego las memorias serán de tamaño 21. En ambos casos la información está más concentrada luego no es necesario considerar un tamaño de memoria tan grande como para el formato frase.

Primero se analizarán los resultados de los modelos por separado para comparar como influye el tipo de representación en el buen funcionamiento de cada uno, igual que se hizo para el estudio de los parámetros. Después se enfrentarán los dos modelos para cada formato considerado. Las gráficas de la figura 4.37 muestran esta primera comparación.

Parámetros óptimos para MemN2N				
	Frase	**Ventana**	**Tripleta**	**Frase + tripleta**
Número de épocas	200			
Número de saltos	3			
Inicialización de los pesos	Normal			
Algoritmo de optimización	SGD			
Tasa de aprendizaje	0.001			
Reducción de la tasa de aprendizaje	-			
Norma del gradiente máxima	40			
Tamaño de inmersión	40			
Tamaño de batch	10			
Tamaño de memoria	75	52	21	75

Tabla 4.38: Selección óptima de parámetros en función del formato con el modelo *MemN2N*.

Parámetros óptimos para KVMemNN				
	Frase	**Ventana**	**Tripleta**	**Frase + tripleta**
Número de épocas	200			
Número de saltos	3			
Inicialización de los pesos	Xavier normal			
Algoritmo de optimización	Adam			
Tasa de aprendizaje	0.001			
Norma del gradiente máxima	40			
Tamaño de características	20			
Tamaño de inmersión	40			
Tamaño de batch	10			
Tamaño de memoria	75	52	21	75

Tabla 4.39: Selección óptima de parámetros en función del formato con el modelo *KV-MemNN*.

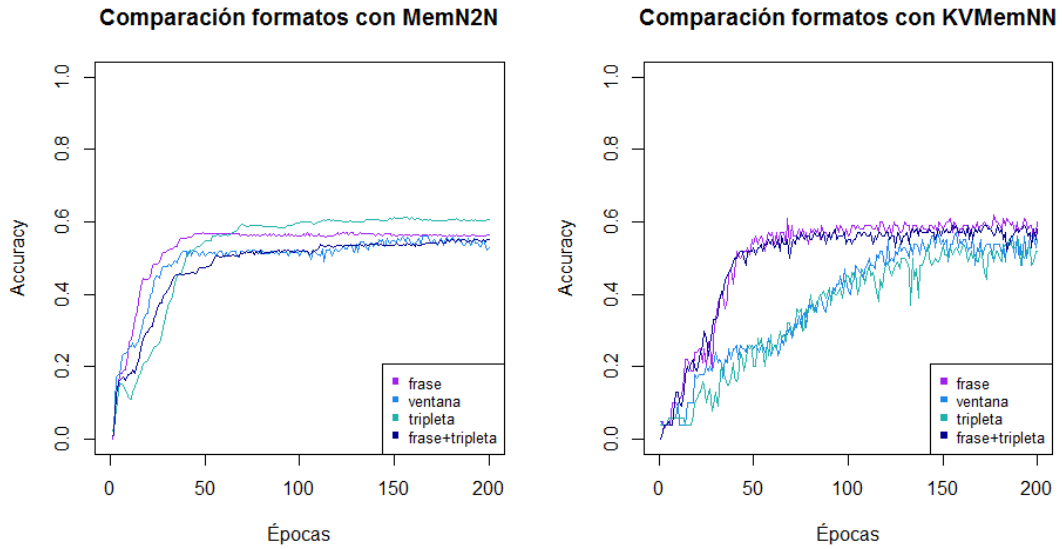


Figura 4.37: Resultados en función del tipo de formato.

A la vista de los resultados sobre el modelo *MemN2N*, se ve que este trabaja mejor con los datos en forma de tripletas, ya que la información está más concentrada. Para el resto de representaciones la red alcanza prácticamente los mismos resultados, siendo un poco mejores los obtenidos para el formato frase.

Por otro lado, teniendo en cuenta ahora los resultados sobre el modelo *KV-MemNN*, se deduce que este es más conveniente cuando se representa la información mediante formato frase. Además, si se considera parte de la información en forma de frases y otra parte en forma de tripletas, los resultados obtenidos son prácticamente iguales, al contrario de lo que pasaba en el otro modelo, lo que resalta la conveniencia de considerar este para trabajar datos en forma de texto completo. Por otro lado, también es prácticamente idéntico el acierto conseguido por la red para información tanto en formato ventana como tripleta.

Dado que interesa saber cuál es el modelo más adecuado para el tratamiento de cada tipo de representación de la información, en la figura 4.38 se analiza la capacidad de aprendizaje de las dos redes dependiendo del formato de los datos.

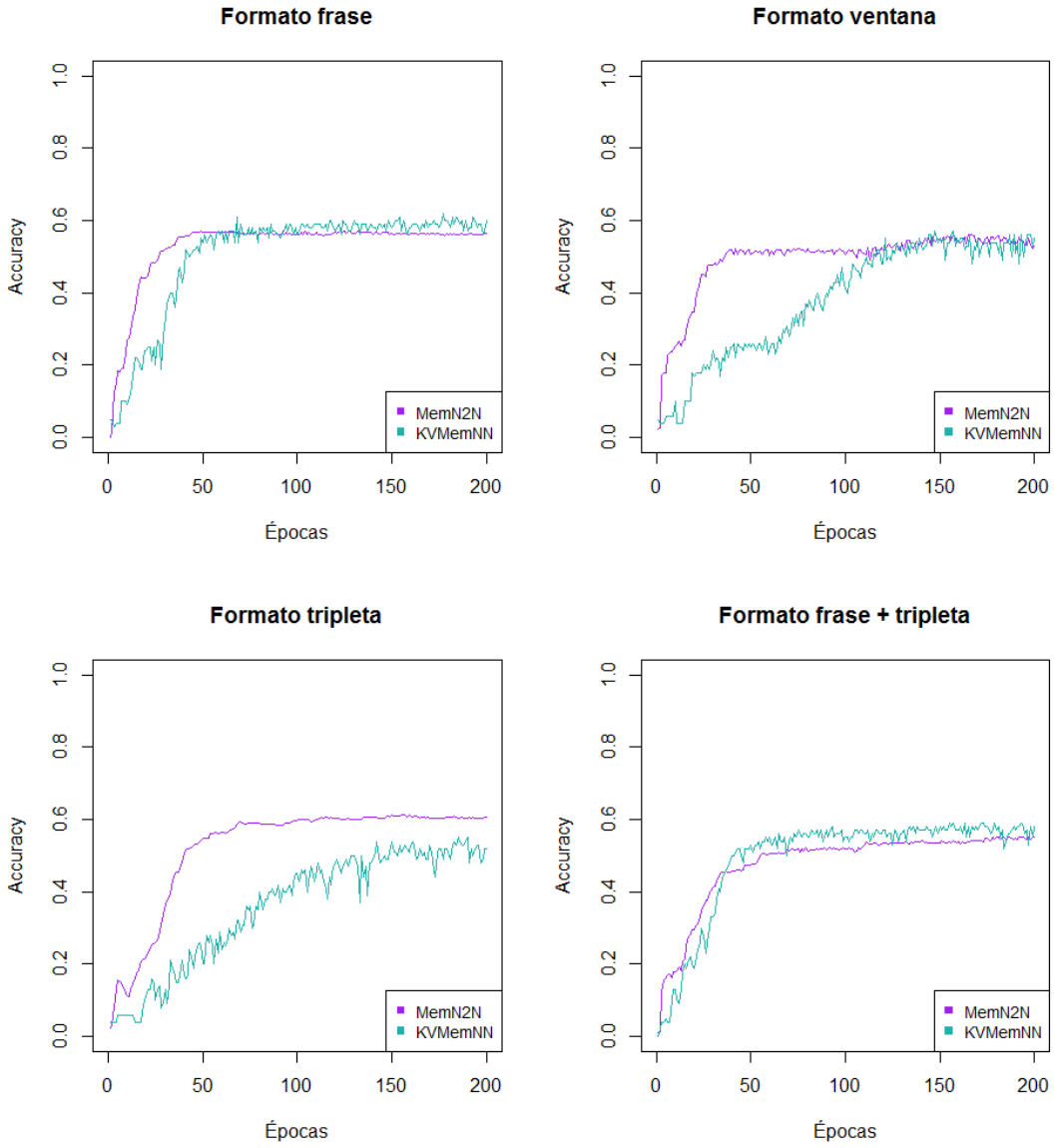


Figura 4.38: Resultados en función del modelo y del tipo de formato.

En cuanto a la representación en forma de frases, los resultados obtenidos por ambos modelos son muy similares, aunque el *KV-MemNN* consigue sobrepasar al *MemN2N* ligeramente. Esto se debe a que cuando se utilizaba este formato en el caso del *KV-MemNN*, la codificación de la frase completa se almacenaba por igual en ambas memorias, lo que hacía a este modelo equivalente al anterior.

Considerando la información en forma de ventanas, los dos modelos alcanzan prácticamente el mismo resultado, mejorando el modelo *KV-MemNN* en las últimas épocas, aunque su convergencia haya sido más lenta.

Si en lugar de disponer de un texto lo que se tiene es una base de conocimiento, es decir la información en forma de tripletas, el modelo *MemN2N* alcanza mejores resultados en el número de épocas considerado. Esto se debe a que la información en forma de tripletas es mucho más reducida, y en el caso del *KV-MemNN* aún se reduce más al guardar en la memoria clave el sujeto y el predicado, y en la memoria valor el objeto únicamente.

Realizar los entrenamientos combinando información en frases y en tripletas hace que la pequeña diferencia que se producía entre los resultados obtenidos por ambos modelos, cuando solo se disponía de información en el formato frase, aumente. Además, la velocidad de convergencia del modelo clave-valor también crece. Esto crea una diferencia en la capacidad de las redes para trabajar con diferentes tipos de información simultáneamente. El modelo *MemN2N* presenta algún problema a la hora de afrontar esta situación, ya que los resultados empeoran con respecto a los obtenidos para los formatos frase y tripleta por separado. Por el contrario, el modelo *KV-MemNN* sí consigue tratar la información combinada. Los resultados que consigue en este caso alcanzan prácticamente a los obtenidos cuando trabajaba sólo con frases completas, por lo que la idea de complementar esta disposición de la información con otra simplificada, como son las tripletas, es una opción a considerar.

Para más detalle, en la tabla 4.40 aparecen los resultados de acierto tras evaluar las redes, una vez han sido entrenadas para cada uno de los formatos. Tras analizar estos resultados, excepto cuando la información se presenta en tripletas, el modelo *KV-MemNN* supera al *MemN2N*.

	Red de Memoria End-to-End	*Red de Memoria Clave-Valor*
Formato	**Accuracy en test**	**Accuracy en test**
Frase	0.56	0.60
Ventana	0.53	0.55
Tripleta	0.61	0.50
Frase + tripleta	0.53	0.58

Tabla 4.40: Resultados test óptimos para cada modelo y formato.

Por último, van a analizarse y compararse los tiempos de procesamiento y entrenamiento necesarios para cada modelo según el tipo de formato, teniendo en cuenta la selección óptima de parámetros en cada caso. Se muestran a continuación en las tablas 4.41 y 4.42. Nótese que la consideración de memoria clave y memoria valor influye en estos tiempos de ejecución.

Comparaciones con modelo MemN2N		
Formato	**Tiempo procesamiento**	**Tiempo entrenamiento**
Frase	1707.98 segs - 28.47 mins	1031.79 segs - 17.20 mins
Ventana	2029.85 segs - 33.83 mins	921.13 segs - 15.35 mins
Tripleta	1344.63 segs - 22.41 mins	827.35 segs - 13.79 mins
Frase + tripleta	1626.60 segs - 27.11 mins	1003.82 segs - 16.73 mins

Tabla 4.41: Tiempos de procesamiento y entrenamiento para los diferentes formatos de la información.

Comparaciones con modelo KVMemNN		
Formato	**Tiempo procesamiento**	**Tiempo entrenamiento**
Frase	1543.31 segs - 25.72 mins	1174.17 segs - 19.57 mins
Ventana	1972.57 segs - 32.88 mins	853.59 segs - 14.23 mins
Tripleta	1242.47 segs - 20.71 mins	755.64 segs - 12.59 mins
Frase + tripleta	1553.21 segs - 25.87 mins	1165.90 segs - 19.43 mins

Tabla 4.42: Tiempos de procesamiento y entrenamiento para los diferentes formatos de la información.

A diferencia del conjunto de datos *tareas bAbI*, para el cual los tiempos de procesamiento de las diferentes tareas eran bastante similares y donde existían diferencias era en los tiempos de entrenamiento, para este conjunto las diferencias se presentan en los tiempos de procesamiento, que están influenciados por la complejidad y dimensión de los datos. Estos tiempos son menores para el modelo *KV-MemNN*. En cuanto a los tiempos de entrenamiento, para los formatos frase y frase-tripleta el modelo *MemN2N* tarda en entrenarse algunos minutos menos, pero para los formatos ventana y tripleta, es el modelo *KV-MemNN* el que tarda menos tiempo, aunque todas estas diferencias son mínimas.

Para ambos modelos, tarda más en procesarse la información en forma de ventanas, lo cual tiene sentido ya que tras analizar los datos deben buscarse las palabras consideradas como entidades para así crear las diferentes ventanas para cada película, antes de proceder a la vectorización de la información. Por el contrario, el formato que tarda menos en procesarse es el tripleta, ya que la información está más condensada y es más rápida de procesar.

En cuanto al formato frase-tripleta, ambos modelos tardan prácticamente lo mismo en procesar la información y vectorizarla que cuando esta se encontraba en forma de frases. únicamente.

Concluir que, dependiendo del formato en que se presente la información que quiera tratarse, interesará la utilización de uno u otro modelo, aunque ambos dos consiguen resultados similares. Concretamente, cuando todo el conjunto de datos se encuentra total o parcialmente en forma de frases con estructura sintáctica compleja, será más conveniente utilizar el modelo *KV-MemNN*, y para trabajar con información simplificada en forma de tripletas, el modelo *MemN2N*.

Capítulo 5

Conclusiones y trabajo futuro

> *« Uno nunca se da cuenta de lo que se ha hecho, uno solo puede ver lo que queda por hacer. »*
> — *Marie Curie*

Finalmente, en este último capítulo, se van a comentar las conclusiones obtenidas tras la realización de este trabajo y posibles cuestiones pendientes que podrían desarrollarse en un futuro.

5.1. Conclusiones

Dado que el principal objetivo era utilizar modelos de Deep Learning para la comprensión de textos, se comenzaron estudiando como base las redes neuronales recurrentes profundas, ya que entre los posibles modelos que abarca el Deep Learning, estas redes son más adecuadas para el procesamiento secuencial de datos presentados como lenguaje natural escrito. Tras trabajar con estas redes y ver que su capacidad de almacenamiento de información es en algunos casos insuficiente, se procedió a estudiar otro tipo de redes que fueron creadas con la idea de solventar el problema de memoria restringida que presentan las redes recurrentes. Estas son las denominadas redes de memoria, que han tenido mayor peso durante el trabajo ya que son más adecuadas para el tipo de análisis que se pretendía hacer y los resultados que querían obtenerse. El estudio llevado a cabo para ambos tipos de redes ha sido tanto a nivel teórico como práctico.

Se ha realizado el estudio desde un punto de vista analítico y de optimización que permitiese conocer a fondo el funcionamiento de cada una de las redes para así poder obtener el mayor rendimiento de las mismas. Para analizar el ajuste de cada red se construyó una herramienta que permitiese visualizar los resultados sobre el conjunto de validación, distinto del de generación del modelo. Tras este análisis, se concluyó que los parámetros que más deben tenerse en cuenta para obtener el mejor modelo son el número de épocas y saltos que realiza la red, la tasa de aprendizaje según cual sea el algoritmo de optimización y actualización de los pesos considerado, y el tamaño que deben de tener sus memorias. Que los parámetros influyan de manera tan considerable en la capacidad de aprendizaje del modelo y por tanto en la obtención de mejores o peores resultados, permite concluir que el aprendizaje automatizado como tal no existe.

Sin embargo, pretender optimizar el funcionamiento de las redes a partir de los parámetros es en vano si no se realiza, previo al entrenamiento, un procesamiento y codificación de los datos adecuado al problema en particular. El correcto tratamiento de la información es lo que marca la diferencia a la hora de obtener buenos resultados. En este sentido se ha visto que para los entrenamientos del modelo *KV-MemNN* sobre el conjunto de datos *WikiMovies* ha sido necesario implementar diferentes funciones

que procesen los textos en función de su formato, ya que la forma de analizar y codificar la información no es igual. Cuando los datos de entrada de la red son frases completas, estos se guardan directamente en vectores. Sin embargo, cuando son tripletas, las palabras que las componen se separan para guardarse en una lista formada por un vector y una palabra. Esta diferencia se debe a que la manera de almacenar los datos también varía al considerar dos memorias con diferente finalidad: recordar que la memoria clave se diseñaba de modo que se relacionase con la pregunta y la memoria valor de acuerdo a la respuesta.

En este sentido, tras los análisis llevados a cabo en función de la disposición de la información, se concluye que debe tenerse en cuenta la estructura de los datos a la hora de elegir los modelos a utilizar, ya que de esto depende también que un modelo tenga ventaja sobre otro. Por ejemplo, en caso de disponer de una base de conocimiento sobre la que se quiera extraer información, lo más conveniente sería trabajar con la red *MemN2N*. Por el contrario, si no se tiene disponible una base de conocimiento o esta es limitada y los datos tienen una estructura sintáctica compleja, debería optarse por la red *KV-MemNN*.

5.2. Limitaciones

Los modelos *MemN2N* y *KV-MemNN*, utilizados para la comparación de los distintos formatos en los que puede presentarse la información textual, se entrenaron sobre parte del conjunto de datos original a causa del problema que presentaba el tiempo computacional. Aunque los resultados obtenidos sobre este subconjunto son suficientes para probar que el modelo *KV-MemNN* funciona mejor que el *MemN2N* cuando se dispone de textos complejos, tanto en porcentaje de acierto como en tiempo computacional, considerar un subconjunto de mayor tamaño o incluso el conjunto de datos completo permitiría obtener mejores resultados al tener más información sobre la que los modelos puedan aprender.

5.3. Futuras líneas de trabajo y aplicaciones

Entrenar los modelos con conjuntos de datos de mayores dimensiones sería la primera línea a seguir. Además, dado que trabajar con la información en tripletas ha funcionado bien, se puede ampliar al estudio con *ontologías* para mejorar la funcionalidad de las redes. Una ontología es una especificación explícita y formal de una conceptualización [23]. Permiten describir el conocimiento que va a manejar un sistema informático, expresado en un cierto lenguaje de representación del conocimiento [24].

En resumen, la idea es trabajar con textos de determinada índole que tengan asociada una base de conocimiento que aporte información extra sobre ellos. Al ser una base de conocimiento, las ontologías tienen una estructura de tripleta, representada mediante nodos conectados. Es por esto por lo que a la hora de estudiar los diferentes modos de representación de la información, se consideró que las redes pudieran tratar también el caso en que los datos se presentasen tanto como frases como en forma de tripletas. Con el análisis realizado se probó que es un planteamiento que puede funcionar.

Dado que existen numerosas ontologías ya creadas y disponibles en la Web; por ejemplo sobre redes semánticas multilingües, sobre investigaciones científicas y médicas, genomas y enfermedades, documentos y publicaciones, etc; esta línea de trabajo puede extenderse a muchos campos.

En cuanto a aplicaciones, la principal aplicación de estos modelos es mejorar los sistemas de búsqueda semántica y mejorar las técnicas de extracción de información. Por ejemplo, a partir de una base de conocimiento médica, hacer consultas en lenguaje natural (en lugar de consultas básicas a bases de datos) con diferentes síntomas que pueda tener un paciente y que el modelo responda con la posible enfermedad que presente y hacer sugerencias de posibles medicamentos para el tratamiento.

Otra aplicación podría ser, a partir de una base de conocimiento con información sobre una empresa: su estructura, formación de trabajadores, servicios que ofrecen, etc.; realizar consultas sobre necesidades particulares que se tengan, como puede ser la utilización de un nuevo software. A partir de todos los datos disponibles sobre la especialización de los trabajadores (a partir de su Curriculum Vitae, cursos que han realizado, proyectos en los que han participado,...) el modelo sugeriría con qué persona debería contactarse para ayudar con esa nueva necesidad.

Como complemento a estas aplicaciones, estaría la mejora de la capacidad de razonamiento propia de los *chatbots* o asistentes, y la ampliación de su conocimiento por medio de las ontologías, ya que permiten inferir nuevo conocimiento a partir del contexto considerado.

Bibliografía

[1] A. GIBSON, J. PATTERSON *Deep Learning (2017)*. Fecha de consulta: noviembre de 2017, https://www.safaribooksonline.com/library/view/deep-learning/9781491924570/.

[2] Y. GOLBERG *A Primer on Neural Network Models for Natural Language Processing (2016)*. Fecha de consulta: noviembre de 2017, https://www.jair.org/media/4992/live-4992-9623-jair.pdf.

[3] M. BODÉN, *A guide to recurrent neural networks and backpropagation (2001)*. Fecha de consulta: septiembre de 2017, https://www.researchgate.net/profile/Mikael_Boden/publication/2903062_A_Guide_to_Recurrent_Neural_Networks_and_Backpropagation/links/53edd2c90cf23733e80b091f/A-Guide-to-Recurrent-Neural-Networks-and-Backpropagation.pdf.

[4] S. HOCHREITER, J. SCHMIDHUBER, *Long Short-Term Memory (1998)*. Fecha de consulta: agosto de 2017, *Neural Computation, volumen 9, edición 8, p. 1735-1780*.

[5] J. WESTON, S. CHOPRA Y A. BORDES, *Memory Networks (2015)*. Fecha de consulta: noviembre de 2017, https://arxiv.org/pdf/1410.3916.pdf.

[6] S. SUKHBAATAR, A. SZLAM, J. WESTON Y R. FERGUS, *End-To-End Memory Networks (2015)*. Fecha de consulta: noviembre de 2017, https://pdfs.semanticscholar.org/d065/727aa583c9651078c9bc53235ef4b5734748.pdf?_ga=2.131213071.1607204614.1493731642-1531649315.1491390290.

[7] A.H. MILLER, A. FISCH, J. DODGE, A-H. KARIMI, A. BORDES Y J. WESTON, *Key-Value Memory Networks for Directly Reading Documents (2016)*. Fecha de consulta: noviembre de 2017, https://arxiv.org/pdf/1606.03126.pdf.

[8] *R: The R Project for Statistical Computing (1993)*. Fecha de consulta: octubre de 2017, https://www.r-project.org/.

[9] *Python (1991)*. Fecha de consulta: octubre de 2017, https://www.python.org/.

[10] GOOGLE, *TensorFlow (2015)*. Fecha de consulta: noviembre de 2017, https://www.tensorflow.org.

[11] *Keras: The Python Deep Learning library*. Fecha de consulta: noviembre de 2017, https://keras.io/.

[12] PONTIFICIA UNIVERSIDAD CATÓLICA DE RÍO DE JANEIRO, *Lua (1993)*. Fecha de consulta: junio de 2017, https://es.wikipedia.org/wiki/Lua.

[13] J. WESTON, A. BORDES, S. CHOPRA, A.M. RUSH, B. VAN MERRIËNBOER, A. JOULIN Y T. MIKOLOV, *Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks (2016)*. Fecha de consulta: noviembre de 2017, https://arxiv.org/pdf/1502.05698.pdf.

[14] FACEBOOK RESEARCH, *bAbI tasks (2015)*. Fecha de consulta: mayo de 2017, https://github.com/facebook/bAbI-tasks.

[15] FACEBOOK RESEARCH, *The bAbI project*. Fecha de consulta: mayo de 2017, https://research.fb.com/downloads/babi/.

[16] R. AZNAR, *Minería de Textos, Trabajo Fin de Máster (2016)*. Fecha de consulta: noviembre de 2017, https://deposita.unizar.es/TAZ/CIEN/2016/31944/TAZ-TFM-2016-1123.pdf

[17] A. GRAVES, *Generating Sequences With Recurrent Neural Networks (2014)*. Fecha de consulta: julio de 2017, https://arxiv.org/pdf/1308.0850.pdf.

[18] W. J. MURDOCH, A. SZLAM, *Automatic Rule Extraction from Long Short Term Memory Networks (2017)*. Fecha de consulta: julio de 2017, https://arxiv.org/pdf/1702.02540.pdf.

[19] O. VINYALS, L. KAISER, T. KOO, S. PETROV, I. SUTSKEVER, G. HINTON, *Grammar as a Foreign Language (2015)*. Fecha de consulta: junio de 2017, https://arxiv.org/pdf/1412.7449.pdf.

[20] WIKIPEDIA, *Gated Recurrent Unit*. Fecha de consulta: julio de 2017. https://en.wikipedia.org/wiki/Gated_recurrent_unit.

[21] D.P. KINGMA, J. LEI BA, *Adam: A Method for Stochastic Optimization (2015)*. Fecha de consulta: noviembre de 2017, https://arxiv.org/pdf/1412.6980.pdf.

[22] X. GLOROT, Y. BENGIO, *Understanding the difficulty of training deep feedforward neural networks (2010)*. Fecha de consulta: noviembre de 2017, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.207.2059&rep=rep1&type=pdf.

[23] T. R. GRUBER, *Toward principles for the design of ontologies used for knowledge sharing*, International Journal of Human and Computer Studies (1995), vol. 43, pp. 907-928. Fecha de consulta: noviembre de 2017.

[24] N.F. NOY Y D.L. MCGUINNESS, *Ontology Development 101: A Guide to Creating Your First Ontology*, Stanford University. Fecha de consulta: noviembre de 2017. http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html.

Anexos

Anexo A

Código Python

En este anexo se muestra parte del código implementado para el análisis de la información y entrenamiento del modelo. La primera sección recoge el código principal del modelo *KV-MemNN*. En las siguientes secciones se exponen las funciones definidas para procesar y vectorizar cada tipo de formato en que puede encontrarse la información.

A.1. Código principal

```
print("Comienzo ejecucion sobre Peliculas:")

starting_point2 = time.time()


train_questions, val_questions, test_questions, entities =
    load_movies_questions(FLAGS.data_dir)

questions = train_questions + val_questions + test_questions


vocab2 = sorted(reduce(lambda x, y: x | y, (set(list(a.rsplit(', ') + q)) for q, a in
    questions)))
vocab5 = sorted(reduce(lambda x, y: x | y, (set(list([a.lstrip()])) for q, a in
    questions)))

vocab = sorted(list(set(vocab2+vocab5+entities)))
word_idx = dict((c, i + 1) for i, c in enumerate(vocab))
vocab_size = len(word_idx) + 1 # +1 for nil word


query_size = max(map(len, (q for q, _ in questions)))



# cargar conjuntos train/val/test segun el tipo de formato del texto

if FLAGS.level == "sentence":

    data = load_movies_sentences(FLAGS.data_dir)


    vocab1 = sorted(reduce(lambda x, y: x | y, (set(list(chain.from_iterable(s))) for
        s in data)))
```

```
vocab_sentence = sorted(list(set(vocab1+vocab)))
word_idx = dict((c, i + 1) for i, c in enumerate(vocab_sentence))
vocab_size = len(word_idx) + 1 # +1 for nil word


max_story_size = max(map(len, (s for s in data)))
mean_story_size = int(np.mean(map(len, (s for s in data))))
sentence_size = max(map(len, chain.from_iterable(s for s in data)))
memory_size = min(FLAGS.memory_size, max_story_size)
#memory_size = min(FLAGS.memory_size, mean_story_size)
sentence_size = max(query_size, sentence_size)

print("Longitud de frase máxima", sentence_size)
print("Longitud de historia máxima", max_story_size)
print("Longitud de historia media", mean_story_size)



S, Q, A = vectorize_todo(data, train_questions, entities, word_idx,
    sentence_size, memory_size)

if Q.shape[0]-S.shape[0] > 0:
    add = np.zeros((Q.shape[0]-S.shape[0],S.shape[1],S.shape[2]), np.int32)
    S = np.concatenate((S, add), axis=0)

trainS = S
trainQ = Q
trainA = A

trainK = trainS
trainV = trainS


S, Q, A = vectorize_todo(data, val_questions, entities, word_idx, sentence_size,
    memory_size)

if Q.shape[0]-S.shape[0] > 0:
    add = np.zeros((Q.shape[0]-S.shape[0],S.shape[1],S.shape[2]), np.int32)
    S = np.concatenate((S, add), axis=0)

valS = S
valQ = Q
valA = A

valK = valS
valV = valS


S, Q, A = vectorize_todo(data, test_questions, entities, word_idx, sentence_size,
    memory_size)

if Q.shape[0]-S.shape[0] > 0:
    add = np.zeros((Q.shape[0]-S.shape[0],S.shape[1],S.shape[2]), np.int32)
    S = np.concatenate((S, add), axis=0)

testS = S
testQ = Q
```

```
    testA = A

    testK = testS
    testV = testS


    #print("VOCABULARIO ", word_idx)


    print("Training set shape", trainS.shape)

    # parámetros
    n_train = trainS.shape[0]
    n_test = testS.shape[0]
    n_val = valS.shape[0]

    print("Tamaño training", n_train)
    print("Tamaño validation", n_val)
    print("Tamaño testing", n_test)



if FLAGS.level == "window":

    windows = load_movies_windows(FLAGS.data_dir)


    vocab4 = sorted(reduce(lambda x, y: x | y,
        (set(list(chain.from_iterable(chain.from_iterable(([a[0]]) for a in s) )))
        for s in windows)))
    vocab44 = sorted(reduce(lambda x, y: x | y,
        (set(list(chain.from_iterable(([a[1]]) for a in s) )) for s in windows)))

    vocab_window = sorted(list(set(vocab+vocab4+vocab44)))
    word_idx = dict((c, i + 1) for i, c in enumerate(vocab_window))
    vocab_size = len(word_idx) + 1 # +1 for nil word


    max_story_window_size = max(map(len, (s for s in windows)))
    mean_story_window_size = int(np.mean(map(len, (s for s in windows))))
    window_size = max(map( len, chain.from_iterable(chain.from_iterable(([a[0]]) for
        a in s) for s in windows)))
    sentence_size = max(query_size, window_size) # for the position
    memory_size = min(FLAGS.memory_size, max_story_window_size)
    #memory_size = mean_story_window_size
    max_window_size = window_size
    mean_window_size = int(np.mean(map(len, chain.from_iterable(chain.from_iterable(w
        for w in s[0]) for s in windows))))

    print("Longitud de historia máxima", max_story_window_size)
    print("Longitud de historia media", mean_story_window_size)
    print("Longitud de frase máxima", sentence_size)
    print("Tamaño máximo de ventana", max_window_size)
    print("Tamaño medio de ventana", mean_window_size)
```

```
K, V, Q, A = vectorize_window_todo(windows, train_questions, entities, word_idx,
    sentence_size, window_size, memory_size)

if Q.shape[0]-K.shape[0] > 0:
    add = np.zeros((Q.shape[0]-K.shape[0],K.shape[1],K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=0)
    add = np.zeros((Q.shape[0]-V.shape[0],V.shape[1],V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=0)

if Q.shape[1]-K.shape[2] > 0:
    add = np.zeros((K.shape[0],K.shape[1],Q.shape[1]-K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=2)
    add = np.zeros((V.shape[0],V.shape[1],Q.shape[1]-V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=2)

trainK = K
trainV = V
trainQ = Q
trainA = A


K, V, Q, A = vectorize_window_todo(windows, val_questions, entities, word_idx,
    sentence_size, window_size, memory_size)

if Q.shape[0]-K.shape[0] > 0:
    add = np.zeros((Q.shape[0]-K.shape[0],K.shape[1],K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=0)
    add = np.zeros((Q.shape[0]-V.shape[0],V.shape[1],V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=0)

if Q.shape[1]-K.shape[2] > 0:
    add = np.zeros((K.shape[0],K.shape[1],Q.shape[1]-K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=2)
    add = np.zeros((V.shape[0],V.shape[1],Q.shape[1]-V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=2)

valK = K
valV = V
valQ = Q
valA = A


K, V, Q, A = vectorize_window_todo(windows, test_questions, entities, word_idx,
    sentence_size, window_size, memory_size)

if Q.shape[0]-K.shape[0] > 0:
    add = np.zeros((Q.shape[0]-K.shape[0],K.shape[1],K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=0)
    add = np.zeros((Q.shape[0]-V.shape[0],V.shape[1],V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=0)

if Q.shape[1]-K.shape[2] > 0:
    add = np.zeros((K.shape[0],K.shape[1],Q.shape[1]-K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=2)
    add = np.zeros((V.shape[0],V.shape[1],Q.shape[1]-V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=2)

testK = K
```

```
    testV = V
    testQ = Q
    testA = A


    #print("VOCABULARIO ", word_idx)


    print("Training set shape", trainK.shape)

    # parámetros
    n_train = trainK.shape[0]
    n_test = testK.shape[0]
    n_val = valK.shape[0]

    print("Training Size", n_train)
    print("Validation Size", n_val)
    print("Testing Size", n_test)



if FLAGS.level == "triple":

    triples = load_movies_triples(FLAGS.data_dir)


    vocab3 = sorted(reduce(lambda x, y: x | y,
        (set(list(chain.from_iterable(chain.from_iterable(([a[0]]) for a in s) )))
        for s in triples)))
    vocab6 = sorted(reduce(lambda x, y: x | y, (set(list(chain.from_iterable(([a[1]])
        for a in s) )) for s in triples)))

    vocab_triple = sorted(list(set(vocab+vocab3+vocab6)))
    word_idx = dict((c, i + 1) for i, c in enumerate(vocab_triple))
    vocab_size = len(word_idx) + 1 # +1 for nil word


    max_story_triple_size = max(map(len, (s for s in triples)))
    mean_story_triple_size = int(np.mean(map(len, (s for s in triples))))
    sentence_size = max(map(len, chain.from_iterable(s for s in triples)))
    sentence_size = max(query_size, sentence_size) # for the position
    memory_size = min(FLAGS.memory_size, max_story_triple_size)
    #memory_size = min(FLAGS.memory_size, mean_story_triple_size)

    print("Longitud de historia máxima", max_story_triple_size)
    print("Longitud de historia media", mean_story_triple_size)



    K, V, Q, A = vectorize_triple_todo(triples, train_questions, entities, word_idx,
        sentence_size, memory_size)

    if Q.shape[0]-K.shape[0] > 0:
        add = np.zeros((Q.shape[0]-K.shape[0],K.shape[1],K.shape[2]), np.int32)
        K = np.concatenate((K, add), axis=0)
        add = np.zeros((Q.shape[0]-V.shape[0],V.shape[1],V.shape[2]), np.int32)
        V = np.concatenate((V, add), axis=0)
```

```
if Q.shape[1]-K.shape[2] > 0:
    add = np.zeros((K.shape[0],K.shape[1],Q.shape[1]-K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=2)
    add = np.zeros((V.shape[0],V.shape[1],Q.shape[1]-V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=2)

trainK = K
trainV = V
trainQ = Q
trainA = A


K, V, Q, A = vectorize_triple_todo(triples, val_questions, entities, word_idx,
    sentence_size, memory_size)

if Q.shape[0]-K.shape[0] > 0:
    add = np.zeros((Q.shape[0]-K.shape[0],K.shape[1],K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=0)
    add = np.zeros((Q.shape[0]-V.shape[0],V.shape[1],V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=0)

if Q.shape[1]-K.shape[2] > 0:
    add = np.zeros((K.shape[0],K.shape[1],Q.shape[1]-K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=2)
    add = np.zeros((V.shape[0],V.shape[1],Q.shape[1]-V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=2)

valK = K
valV = V
valQ = Q
valA = A


K, V, Q, A = vectorize_triple_todo(triples, test_questions, entities, word_idx,
    sentence_size, memory_size)

if Q.shape[0]-K.shape[0] > 0:
    add = np.zeros((Q.shape[0]-K.shape[0],K.shape[1],K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=0)
    add = np.zeros((Q.shape[0]-V.shape[0],V.shape[1],V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=0)

if Q.shape[1]-K.shape[2] > 0:
    add = np.zeros((K.shape[0],K.shape[1],Q.shape[1]-K.shape[2]), np.int32)
    K = np.concatenate((K, add), axis=2)
    add = np.zeros((V.shape[0],V.shape[1],Q.shape[1]-V.shape[2]), np.int32)
    V = np.concatenate((V, add), axis=2)

testK = K
testV = V
testQ = Q
testA = A


# parámetros
n_train = trainK.shape[0]
n_test = testK.shape[0]
n_val = valK.shape[0]
```

```
    print("Training Size", n_train)
    print("Validation Size", n_val)
    print("Testing Size", n_test)



if FLAGS.level == "sentencetriple":

    data = load_movies_sentences(FLAGS.data_dir)
    triples = load_movies_triples(FLAGS.data_dir)


    vocab7 = sorted(reduce(lambda x, y: x | y, (set(list(chain.from_iterable(s))) for
        s in data)))

    vocab_sentence = sorted(list(set(vocab7+vocab)))
    word_idx = dict((c, i + 1) for i, c in enumerate(vocab_sentence))

    vocab8 = sorted(reduce(lambda x, y: x | y,
        (set(list(chain.from_iterable(chain.from_iterable(([a[0]]) for a in s) )))
        for s in triples)))
    vocab9 = sorted(reduce(lambda x, y: x | y, (set(list(chain.from_iterable(([a[1]])
        for a in s) )) for s in triples)))

    vocab_triple = sorted(list(set(vocab+vocab7+vocab8+vocab9)))
    word_idx = dict((c, i + 1) for i, c in enumerate(vocab_triple))
    vocab_size = len(word_idx) + 1 # +1 for nil word


    max_story_size = max(map(len, (s for s in data)))
    mean_story_size = int(np.mean(map(len, (s for s in data))))
    sentence_size = max(map(len, chain.from_iterable(s for s in data)))
    sentence_size = max(query_size, sentence_size) # for the position
    max_story_triple_size = max(map(len, (s for s in triples)))
    memory_size = min(FLAGS.memory_size, max_story_size)
    #memory_size = min(FLAGS.memory_size, mean_story_size)

    print("Longitud de frase máxima", sentence_size)
    print("Longitud de historia máxima", max_story_size)
    print("Longitud de historia media", mean_story_size)



    K, V, Q, A = vectorize_sent_triple_todo(data, triples, train_questions, entities,
        word_idx, sentence_size, memory_size)

    trainK = K
    trainV = V
    trainQ = Q
    trainA = A


    K, V, Q, A = vectorize_sent_triple_todo(data, triples, val_questions, entities,
        word_idx, sentence_size, memory_size)

    valK = K
    valV = V
```

```
    valQ = Q
    valA = A


    K, V, Q, A = vectorize_sent_triple_todo(data, triples, test_questions, entities,
        word_idx, sentence_size, memory_size)

    testK = K
    testV = V
    testQ = Q
    testA = A


    # parámetros
    n_train = trainK.shape[0]
    n_test = testK.shape[0]
    n_val = valK.shape[0]

    print("Training Size", n_train)
    print("Validation Size", n_val)
    print("Testing Size", n_test)



elapsed_time2 = time.time() - starting_point2

with open("kvmemnn_tiempo_pelis_" + "_level" + str(FLAGS.level) + "_hops" +
    str(FLAGS.hops) + "_epochs" + str(FLAGS.epochs) + "_lrate" +
    str(FLAGS.learning_rate) + "_arate" + str(FLAGS.lrate_decay_steps) + "_batch" +
    str(FLAGS.batch_size) + "_eps" + str(FLAGS.epsilon) + "_feat" +
    str(FLAGS.feature_size) + "_emb" + str(FLAGS.embedding_size) + "_mem" +
    str(memory_size) + "_gradnorm" + str(FLAGS.max_grad_norm) + ".csv", 'a') as f:
f.write('{}, {}, {}, {}\n'.format("Procesamiento (segs)", "Procesamiento (mins)",
    "Entrenamiento (segs)", "Entrenamiento (mins)"))


train_labels = np.argmax(trainA, axis=1)
test_labels = np.argmax(testA, axis=1)
val_labels = np.argmax(valA, axis=1)

batch_size = FLAGS.batch_size
batches = zip(range(0, n_train-batch_size, batch_size), range(batch_size, n_train,
    batch_size))



with tf.Graph().as_default():
    session_conf = tf.ConfigProto( allow_soft_placement=FLAGS.allow_soft_placement,
        log_device_placement=FLAGS.log_device_placement)

    global_step = tf.Variable(0, name="global_step", trainable=False)
    # decay learning rate
    starter_learning_rate = FLAGS.learning_rate
    decay_steps = FLAGS.lrate_decay_steps

    learning_rate = tf.train.exponential_decay(starter_learning_rate, global_step,
        20000, 0.96, staircase=True)
```

```
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate,
    epsilon=FLAGS.epsilon)
#optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)


with tf.Session() as sess:

    model = MemNN_KV(batch_size=batch_size, vocab_size=vocab_size,
        query_size=sentence_size, story_size=sentence_size,
        memory_key_size=memory_size, feature_size=FLAGS.feature_size,
        memory_value_size=memory_size, embedding_size=FLAGS.embedding_size,
        hops=FLAGS.hops, reader=FLAGS.reader, l2_lambda=FLAGS.l2_lambda)

    grads_and_vars = optimizer.compute_gradients(model.loss_op)
    grads_and_vars = [(tf.clip_by_norm(g, FLAGS.max_grad_norm), v) for g, v in
        grads_and_vars if g is not None]
    grads_and_vars = [(add_gradient_noise(g), v) for g, v in grads_and_vars]
    nil_grads_and_vars = []
    for g, v in grads_and_vars:
    if v.name in model._nil_vars:
        nil_grads_and_vars.append((zero_nil_slot(g), v))
    else:
        nil_grads_and_vars.append((g, v))

    train_op = optimizer.apply_gradients(nil_grads_and_vars, name="train_op",
        global_step=global_step)
    sess.run(tf.initialize_all_variables())

    def train_step(k, v, q, a):
        feed_dict = {
            model._memory_value: v,
            model._query: q,
            model._memory_key: k,
            model._labels: a,
            model.keep_prob: FLAGS.keep_prob
        }
        _, step, predict_op = sess.run([train_op, global_step, model.predict_op],
            feed_dict)
        return predict_op


    def test_step(k, v, q):
        feed_dict = {
            model._query: q,
            model._memory_key: k,
            model._memory_value: v,
            model.keep_prob: 1
        }
        preds = sess.run(model.predict_op, feed_dict)
        return preds


    starting_point1 = time.time()

    for t in range(1, FLAGS.epochs+1):
        np.random.shuffle(batches)
        train_preds = []
```

```
            for start in range(0, n_train, batch_size):
                end = start + batch_size
                k = trainK[start:end]
                v = trainV[start:end]
                q = trainQ[start:end]
                a = trainA[start:end]

                predict_op = train_step(k, v, q, a)
                train_preds += list(predict_op)

            train_acc = metrics.accuracy_score(np.array(train_preds), train_labels)


            print('-----------------------')
            print('Epoch', t)
            print('Training Accuracy: {0:.2f}'.format(train_acc))
            print('-----------------------')


            if t % FLAGS.evaluation_interval == 0:

                # testear en conjunto train
                train_preds = test_step(trainK, trainV, trainQ)
                train_acc = metrics.accuracy_score(train_labels, train_preds)
                train_acc = '{0:.2f}'.format(train_acc)

                # conjunto val
                val_preds = test_step(valK, valV, valQ)
                val_acc = metrics.accuracy_score(val_labels, val_preds)
                val_acc = '{0:.2f}'.format(val_acc)

                # conjunto test
                test_preds = test_step(testK, testV, testQ)
                test_acc = metrics.accuracy_score(test_labels, test_preds)
                test_acc = '{0:.2f}'.format(test_acc)

                with open("kvmemnn_peliculasfinal" + "_level" + str(FLAGS.level) +
                    "_hops" + str(FLAGS.hops) + "_epochs" + str(FLAGS.epochs) +
                    "_lrate" + str(FLAGS.learning_rate) + "_arate" +
                    str(FLAGS.lrate_decay_steps) + "_batch" + str(FLAGS.batch_size) +
                    "_eps" + str(FLAGS.epsilon) + "_feat" + str(FLAGS.feature_size) +
                    "_emb" + str(FLAGS.embedding_size) + "_mem" + str(memory_size) +
                    "_gradnorm" + str(FLAGS.max_grad_norm) + ".csv", 'a') as f:
                    f.write('{}, {}, {}, {}\n'.format(t, train_acc, val_acc, test_acc))


                print('-----------------------')
                print('Epoch', t)
                print('Validation Accuracy:', val_acc)
                print('-----------------------')



        elapsed_time1 = time.time() - starting_point1


with open("kvmemnn_tiempo_pelis" + "_level" + str(FLAGS.level) + "_hops" +
    str(FLAGS.hops) + "_epochs" + str(FLAGS.epochs) + "_lrate" +
```

```
    str(FLAGS.learning_rate) + "_arate" + str(FLAGS.lrate_decay_steps) + "_batch" +
    str(FLAGS.batch_size) + "_eps" + str(FLAGS.epsilon) + "_feat" +
    str(FLAGS.feature_size) + "_emb" + str(FLAGS.embedding_size) + "_mem" +
    str(memory_size) + "_gradnorm" + str(FLAGS.max_grad_norm) + ".csv", 'a') as f:
    f.write('{}, {}, {}, {}\n'.format(elapsed_time2, elapsed_time2/60, elapsed_time1,
        elapsed_time1/60))
```

A.2. Funciones comunes

```
def parse_entities(entities_files):

    ent_list = []
    re_list = []
    entities = {}
    ent_rev = {}

    print('Procesando archivo de entidades...')

    with open(entities_files) as read:
        for l in read:
            l = l.rstrip()
            if len(l) > 0:
                ent_list.append(l)
    ent_list.sort(key=lambda x: -len(x))


    for i in range(len(ent_list)):
        k = ent_list[i]
        v = '__{}__'.format(i)
        entities[k] = v
        ent_rev[v] = k
        re_list = [
            (
                re.compile('\\b{}\\b'.format(re.escape(e))),
                '{}'.format(entities[e])
            ) for e in ent_list
        ]


    return ent_rev, re_list
```

```
def parse_questions(lines, ent_list, ent_rev):
    '''Analisis de las preguntas dadas en formato peliculas
    '''

    print("Analizando preguntas y respuestas")
    data=[]
    story = []
    for line in lines:
        if '\xef\xbb\xbf1' in line:
        line = line.replace("\xef\xbb\xbf1", "1")
```

```
        nid, line = line.split(' ', 1)
        nid = int(nid)
        if nid == 1:
            q, a = line.split('\t')
            q = tokenize_movies(q, ent_list, ent_rev)
            a = a.replace('\r', '')

            for ent, idx in ent_list:
                a = ent.sub(idx, a)
                a = a.replace('\n', '')

            for an in a:
                for k, v in ent_rev.items():
                    if k in a:
                        a = a.replace(k, v)
                        break


            # remove question marks
            if q[-1] == "?":
                q = q[:-1]

            story.append('')
            data.append((q, a))

    return data
```

A.3. Funciones formato frase

```
def parse_arguments(lines, ent_list, ent_rev):
    '''Analisis de las historias dadas en formato peliculas
    '''

    print("Analizando peliculas")

    data = []
    story = []
    for line in lines:
        #line = line.decode('utf-8').strip()
        if '\xef\xbb\xbf1' in line:
            line = line.replace("\xef\xbb\xbf1", "1")

        if len(line)>2:
            nid, line = line.split(' ', 1)
            nid = int(nid)
        if nid == 1:
            story = []

        if len(line)<2:
            data.append(story)

        if len(line)>2:
            sent = tokenize_movies(line, ent_list, ent_rev)
            story.append(sent)
```

```
    return data
```

```
def vectorize_todo(data, questions, entities, word_idx, sentence_size, memory_size):

    S = [] # Historia
    Q = [] # Pregunta
    A = [] # Respuesta


    print("Vectorizando peliculas")

    for query, answer in questions:
        ss = []
        lq = max(0, sentence_size - len(query))
        q = [word_idx[w] for w in query] + [0] * lq

        y = np.zeros(len(word_idx) + 1) # 0 is reserved for nil word
        y[word_idx[answer]] = 1


        ent_in_questions = []

        for ent in entities:
            if ent in query or ent in answer:
                ent_in_questions.append(ent)

            ent_in_questions = sorted(reduce(lambda x, y: x | y, (set(list([ent])) for
                ent in ent_in_questions)))




        for story in data:
            story_corresp=[]

            for word in story[0]:
                if word in ent_in_questions and word!= "film":
                    story_corresp = story


            for i, sentence in enumerate(story_corresp, 1):
                ls = max(0, sentence_size - len(sentence))
                ss.append([word_idx[w] for w in sentence] + [0] * ls)


        # guardar solo las frases que caben en la memoria
        ss = ss[::-1][:memory_size]


        # rellenar hasta el tamaño de memoria
        lm = max(0, memory_size - len(ss))
        for _ in range(lm):
            ss.append([0] * sentence_size)
```

```
        S.append(ss)
        Q.append(q)
        A.append(y)

    return np.array(S), np.array(Q), np.array(A)
```

```
def tokenize_movies(sent, entities_list, ent_rev):
    for ent, idx in entities_list:
        sent = ent.sub(idx, sent)

    sent = [x.strip() for x in re.split('(\W+)?', sent) if x.strip()]


    for idx, s in enumerate(sent):
        for k, v in ent_rev.items():
            if s == k:
                s = v
                sent[idx] = v


    return sent
```

A.4. Funciones formato tripleta

```
def parse_triples(lines, ent_list, ent_rev):

    print("Analizando tripletas")

    data = []
    story = []


    for line in lines:
        if len(line)>2:
            nid, line = line.split(' ', 1)
        nid = int(nid)
        if nid == 1:
            story = []

        if len(line)<2: # cambio de pelicula
            data.append(story)

        if len(line)>2:
            sent, obj = tokenize_triple(line, ent_list, ent_rev)

            if obj[-1] == ".":
                obj = obj[:-1]

            story.append((sent,obj))

    return data
```

```
def vectorize_triple_todo(data, questions, entities, word_idx, sentence_size,
    memory_size):

    K = [] # Clave
    V = [] # Valor
    Q = [] # Pregunta
    A = [] # Respuesta


    print("Vectorizando peliculas")

    for query, answer in questions:
        ss = []
        dd = []

        lq = max(0, sentence_size - len(query))
        q = [word_idx[w] for w in query] + [0] * lq

        y = np.zeros(len(word_idx) + 1) # 0 is reserved for nil word
        y[word_idx[answer]] = 1


        ent_in_questions = []

        for ent in entities:
            if ent in query or ent in answer:
                ent_in_questions.append(ent)

            ent_in_questions = sorted(reduce(lambda x, y: x | y, (set(list([ent])) for
                ent in ent_in_questions)))


        for story in data:
            story_corresp = []

            for word in story[0][0]:
                if word in ent_in_questions and word!= "film":
                    story_corresp = story

            for key, value in story_corresp:
                value = [value]
                ss.append([word_idx[k] for k in key])
                dd.append([word_idx[v] for v in value])


        # guardar solo las frases que caben en la memoria
        ss = ss[::-1][:memory_size]
        dd = dd[::-1][:memory_size]


        # rellenar hasta el tamaño de memoria
        lm = max(0, memory_size - len(ss))
        ln = max(0, memory_size - len(dd))

        for _ in range(lm):
            ss.append([0] * 2)
```

```
        for _ in range(ln):
            dd.append([0] * 1)


        K.append(ss)
        V.append(dd)
        Q.append(q)
        A.append(y)

    return np.array(K), np.array(V), np.array(Q), np.array(A)
```

```
def tokenize_triple(sent, entities_list, ent_rev):

    for ent, idx in entities_list:
        sent = ent.sub(idx, sent)

    sent, b , c = sent.split(' ', 2)
    sent = sent + ' ' + b
    a = c.replace('\n', '')
    a = a.replace('\r', '')

    sent = [x.strip() for x in re.split('(\W+)?', sent) if x.strip()]

    for k, v in ent_rev.items():
        if sent[0] == k:
            sent[0] = v
            break


    for k, v in ent_rev.items():
        if k in a:
            a = a.replace(k, v)


    return sent, a
```

A.5. Funciones formato ventana

```
def parse_windows(lines, ent_list, ent_rev):
    '''Analisis de las historias dadas en formato pelicuals
    '''

    print("Analizando peliculas")

    data = []
    story = []
    for line in lines:
        #line = line.decode('utf-8').strip()
        if '\xef\xbb\xbf1' in line:
            line = line.replace("\xef\xbb\xbf1", "1")
        if len(line)>2:
            nid, line = line.split(' ', 1)
        nid = int(nid)
```

```
        if nid == 1:
            story = []
            substory = None

        if len(line)<2:
            substory = [x for x in story if x]
            data.append(substory)

        if len(line)>2 and nid!=1:
            sent = tokenize_movies(line, ent_list, ent_rev)
            story.append(sent)


    max_length = None

    flatten = lambda data: reduce(lambda x, y: x + y, data)
    data = [(flatten(story)) for story in data if not max_length or
        len(flatten(story)) < max_length]


    story2 = []
    data2 = []

    for story in data:
        story2 = []

        for v in story:
            for k, vv in ent_rev.items():
                if v == vv:
                    idx = story.index(v)

                    if idx in range(3, len(story)-3):
                        window = [story[idx-3], story[idx-2], story[idx-1], story[idx],
                            story[idx+1], story[idx+2], story[idx+3]]

                        center = v
                        story2.append((window, center))


        data2.append(story2)

    return data2
```

```
def vectorize_window_todo(data, questions, entities, word_idx, sentence_size,
    window_size, memory_size):

    K = [] # Clave
    V = [] # Valor
    Q = [] # Pregunta
    A = [] # Respuesta


    print("Vectorizando peliculas")

    for query, answer in questions:
        ss = []
```

```
        dd = []

        lq = max(0, sentence_size - len(query))
        q = [word_idx[w] for w in query] + [0] * lq

        y = np.zeros(len(word_idx) + 1) # 0 is reserved for nil word
        y[word_idx[answer]] = 1


        ent_in_questions = []

        for ent in entities:
            if ent in query or ent in answer:
                ent_in_questions.append(ent)

            ent_in_questions = sorted(reduce(lambda x, y: x | y, (set(list([ent])) for
                ent in ent_in_questions)))


        for story in data:
            story_corresp = []

            for word in story[0][0]:
                if word in ent_in_questions and word!= "film":
                    story_corresp = story


            for key, value in story_corresp:
                value = [value]
                ss.append([word_idx[k] for k in key])
                dd.append([word_idx[v] for v in value])


        # guardar solo las frases que caben en la memoria
        ss = ss[::-1][:memory_size]
        dd = dd[::-1][:memory_size]


        # rellenar hasta el tamaño de memoria
        lm = max(0, memory_size - len(ss))
        ln = max(0, memory_size - len(dd))

        for _ in range(lm):
            ss.append([0] * window_size)
        for _ in range(ln):
            dd.append([0] * 1)


        K.append(ss)
        V.append(dd)
        Q.append(q)
        A.append(y)

    return np.array(K), np.array(V), np.array(Q), np.array(A)
```

A.6. Funciones formato frase + tripleta

```
def vectorize_sent_triple_todo(data_sent, data_triple, questions, entities, word_idx,
    sentence_size, memory_size):

    K = [] # Clave
    V = [] # Valor
    Q = [] # Pregunta
    A = [] # Respuesta


    print("Vectorizando peliculas")

    for query, answer in questions:
        ss = []
        dd = []
        lq = max(0, sentence_size - len(query))
        q = [word_idx[w] for w in query] + [0] * lq

        y = np.zeros(len(word_idx) + 1) # 0 is reserved for nil word
        y[word_idx[answer]] = 1


        ent_in_questions = []

        for ent in entities:
            if ent in query or ent in answer:
                ent_in_questions.append(ent)

            ent_in_questions = sorted(reduce(lambda x, y: x | y, (set(list([ent])) for
                ent in ent_in_questions)))


        for story in data_sent:
            story_corresp = []

            for word in story[0]:
                if word in ent_in_questions and word!= "film":
                story_corresp = story

            for i, sentence in enumerate(story_corresp, 1):
                ls = max(0, sentence_size - len(sentence))
                ss.append([word_idx[w] for w in sentence] + [0] * ls)
                dd.append([word_idx[w] for w in sentence] + [0] * ls)


        if story_corresp == []:
            for story in data_triple:

                for word in story[0][0]:
                    if word in ent_in_questions and word!= "film":
                    story_corresp = story

                for key, value in story_corresp:
                    value = [value]
                    ls = max(0, sentence_size - len(key))
                    ld = max(0, sentence_size - len(value))
```

```
                ss.append([word_idx[k] for k in key] + [0] * ls)
                dd.append([word_idx[v] for v in value] + [0] * ld)


        # guardar solo las frases que caben en la memoria
        ss = ss[::-1][:memory_size]
        dd = dd[::-1][:memory_size]


        # rellenar hasta el tamaño de memoria
        lm = max(0, memory_size - len(ss))
        ln = max(0, memory_size - len(dd))

        for _ in range(lm):
            ss.append([0] * sentence_size)
        for _ in range(ln):
            dd.append([0] * sentence_size)


        K.append(ss)
        V.append(dd)
        Q.append(q)
        A.append(y)

    return np.array(K), np.array(V), np.array(Q), np.array(A)
```

Anexo B

Código R

Todos los resultados obtenidos de los entrenamientos se han guardado en archivos con formato `.csv` para importarlos de manera sencilla con R. Para esto se ha utilizado la siguiente orden:

```
nombre_conjunto <- read.csv("direccion_archivo/nombre_conjunto.csv",
      header = FALSE, col.names = c("Epocas", "Train", "Valid", "Test"))
```

Se incluye solo, por secciones en función del modelo considerado, el código utilizado para la creación de las diferentes gráficas.

B.1. RNN

tareas bAbI

```
# 60 epocas
plot(conjunto1_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,60), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto5_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")


# comparacion learning rate 0.1 - 0.01 - 0.001

plot(conjunto1_epocas60_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,60), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas60_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas60_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,60), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
```

```
lines(conjunto5_epocas60_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas60_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,60), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas60_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 100 epocas
plot(conjunto1_epocas100_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto5_epocas100_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas100_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")



# 200 epocas
plot(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")



# comparacion 60-100-200 epocas, lrate 0.01

par(mfrow=c(1,3))

plot(conjunto1_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas100_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
```

```
lines(conjunto5_epocas100_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas100_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion 60-100-200 epocas, lrate 0.001

par(mfrow=c(1,3))

plot(conjunto1_epocas60_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas100_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas60_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas100_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas60_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas100_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, lrate 0.01, comparacion decay 0 - 0.001 - 0.0000001
```

```
par(mfrow=c(1,3))

plot(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_decay0.001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_decay0.0000001_batch32_emb50_neurona_LSTM$Tes
    col="lightseagreen")
legend("bottomright",legend=c("reducción tasa de aprendizaje 0","reducción tasa de
    aprendizaje 0.001","reducción tasa de aprendizaje 0.0000001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_decay0.001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_decay0.0000001_batch32_emb50_neurona_LSTM$Tes
    col="lightseagreen")
legend("bottomright",legend=c("reducción tasa de aprendizaje 0","reducción tasa de
    aprendizaje 0.001","reducción tasa de aprendizaje 0.0000001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_decay0.001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_decay0.0000001_batch32_emb50_neurona_LSTM$Te
    col="lightseagreen")
legend("bottomright",legend=c("reducción tasa de aprendizaje 0","reducción tasa de
    aprendizaje 0.001","reducción tasa de aprendizaje 0.0000001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, comparacion ADAM lrate 0.1 - 0.01 - 0.001

par(mfrow=c(1,3))

plot(conjunto1_epocas200_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(conjunto5_epocas200_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
```

```
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas200_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, comparacion SGD lrate 0.1 - 0.01 - 0.001

par(mfrow=c(1,3))

plot(conjunto1_epocas200_SGD_lrate0.1_batch32_emb50_neurona_LSTM$Test, xlim=c(0,200),
    ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1", type="l",
    col="purple")
lines(conjunto1_epocas200_SGD_lrate0.01_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_SGD_lrate0.001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas200_SGD_lrate0.1_batch32_emb50_neurona_LSTM$Test, xlim=c(0,200),
    ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5", type="l",
    col="purple")
lines(conjunto5_epocas200_SGD_lrate0.01_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_SGD_lrate0.001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas200_SGD_lrate0.1_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas200_SGD_lrate0.01_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_SGD_lrate0.001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
# 200 epocas, comparacion ADAM 0.01 - SGD 0.1

par(mfrow=c(1,3))

plot(conjunto1_epocas200_SGD_lrate0.1_batch32_emb50_neurona_LSTM$Test, xlim=c(0,200),
    ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1", type="l",
    col="purple")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("optimizador SGD","optimizador ADAM"),
    pch=15,col=c("purple","lightseagreen"))

plot(conjunto5_epocas200_SGD_lrate0.1_batch32_emb50_neurona_LSTM$Test, xlim=c(0,200),
    ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1", type="l",
    col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("optimizador SGD","optimizador ADAM"),
    pch=15,col=c("purple","lightseagreen"))

plot(conjunto10_epocas200_SGD_lrate0.1_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("optimizador SGD","optimizador ADAM"),
    pch=15,col=c("purple","lightseagreen"))



# 200 epocas, lrate 0.01, comparacion embedding 20-50-80

par(mfrow=c(1,3))

plot(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb20_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 50","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb20_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 50","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
plot(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb20_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 50","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, lrate 0.01, embedding 50, comparacion batch 10-32-50

par(mfrow=c(1,3))

plot(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, lrate 0.01, embedding 80, comparacion batch 10-32-50

par(mfrow=c(1,3))

plot(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb80_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
```

```
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb80_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb80_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb80_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb80_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb80_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, lrate 0.01, embedding 50-80, comparacion batch 10-32-50

par(mfrow=c(1,3))

plot(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb50_neurona_LSTM$Test,
    col="lightseagreen")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb80_neurona_LSTM$Test,
    col="red2")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    col="orange")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb80_neurona_LSTM$Test,
    col="gold")
legend("bottomright",legend=c("inmersión 50","batch 10","batch 32","batch 50"),
    pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("inmersión 80","batch 10","batch 32","batch 50"),
    pch=15,col=c("white","red2","orange","gold"))

plot(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb50_neurona_LSTM$Test,
    col="lightseagreen")
```

```
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb80_neurona_LSTM$Test,
    col="red2")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    col="orange")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb80_neurona_LSTM$Test,
    col="gold")
legend("bottomright",legend=c("inmersión 50","batch 10","batch 32","batch 50"),
    pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("inmersión 80","batch 10","batch 32","batch 50"),
    pch=15,col=c("white","red2","orange","gold"))

plot(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb50_neurona_LSTM$Test,
    col="lightseagreen")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch10_emb80_neurona_LSTM$Test,
    col="red2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    col="orange")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch50_emb80_neurona_LSTM$Test,
    col="gold")
legend("bottomright",legend=c("inmersión 50","batch 10","batch 32","batch 50"),
    pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("inmersión 80","batch 10","batch 32","batch 50"),
    pch=15,col=c("white","red2","orange","gold"))



# 200 epocas, lrate 0.01, embedding 50, batch 32, comparacion neuronas LSTM - GRU -
    RNN

par(mfrow=c(1,3))

plot(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_GRU$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_RNN$Test,
    col="lightseagreen")
legend("bottomright",legend=c("celda LSTM","celda GRU", "celda RNN"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_GRU$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_RNN$Test,
    col="lightseagreen")
legend("bottomright",legend=c("celda LSTM","celda GRU", "celda RNN"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
```

```
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_GRU$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_RNN$Test,
    col="lightseagreen")
legend("bottomright",legend=c("celda LSTM","celda GRU", "celda RNN"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# parametros optimos: 200 epocas, ADAM con lrate 0.01, embedding 80, batch 32, celda
    LSTM - GRU

par(mfrow=c(1,3))

plot(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_GRU$Test,
    col="lightseagreen")
legend("bottomright",legend=c("celda LSTM","celda GRU"),
    pch=15,col=c("purple","lightseagreen"))

plot(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_GRU$Test,
    col="lightseagreen")
legend("bottomright",legend=c("celda LSTM","celda GRU"),
    pch=15,col=c("purple","lightseagreen"))

plot(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb80_neurona_GRU$Test,
    col="lightseagreen")
legend("bottomright",legend=c("celda LSTM","celda GRU"),
    pch=15,col=c("purple","lightseagreen"))



# 60 epocas, lrate 0.1-0.01-0.001, comparacion epsilon 0.1 - 1e-8

par(mfrow=c(1,3))

plot(conjunto10_epocas60_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,60), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas60_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
lines(conjunto10_epocas60_ADAM_lrate0.1_eps0.1_batch32_emb50_neurona_LSTM$Test,
    col="red2")
lines(conjunto10_epocas60_ADAM_lrate0.01_eps0.1_batch32_emb50_neurona_LSTM$Test,
    col="orange")
```

```
lines(conjunto10_epocas60_ADAM_lrate0.001_eps0.1_batch32_emb50_neurona_LSTM$Test,
    col="gold")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion 60-100-200 epocas, lrate 0.01

par(mfrow=c(1,3))

plot(conjunto1_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas100_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas100_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas100_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion 60-100-200 epocas, lrate 0.1

par(mfrow=c(1,3))

plot(conjunto1_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas100_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
plot(conjunto5_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas100_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas100_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.1_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion 60-100-200 epocas, lrate 0.001

par(mfrow=c(1,3))

plot(conjunto1_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas100_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas100_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas60_ADAM_lrate0.01_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas100_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
par(mfrow=c(1,3))

plot(conjunto1_epocas100_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto1_epocas100_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")

plot(conjunto5_epocas100_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(conjunto5_epocas100_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")

plot(conjunto10_epocas100_ADAM_lrate0.001_eps0.00000001_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(conjunto10_epocas100_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")


plot(conjunto1_epocas60_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test, xlim=c(0,200),
    ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1", type="l",
    col="purple")
lines(conjunto1_epocas100_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto1_epocas200_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto5_epocas60_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test, xlim=c(0,200),
    ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1", type="l",
    col="purple")
lines(conjunto5_epocas100_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto5_epocas200_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(conjunto10_epocas60_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(conjunto10_epocas100_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    col="dodgerblue2")
lines(conjunto10_epocas200_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(conjunto1_epocas60_ADAM_ORIGINAL_batch32_emb50_neurona_LSTM$Test, xlim=c(0,60),
    ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1", type="l",
    col="purple")
lines(conjunto1_epocas60_ADAM_ORIGINAL_batch32_emb40_neurona_LSTM$Test,
    col="dodgerblue2")
```

B.2. MemN2N

tareas bAbI

```
#comparacion 60-100-200 epocas

par(mfrow=c(1,3))

plot(memn2n_epocas60_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas200_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas60_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas200_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas60_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas200_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


par(mfrow=c(1,3))

plot(memn2n_epocas60_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas200_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1.1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas60_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas200_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5.1$Test,
    col="lightseagreen")
```

```
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas60_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas200_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10.1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



#comparacion lrate 0.001-0.01-0.1

par(mfrow=c(1,3))

plot(memn2n_epocas100_batch32_lrate0.001_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.1_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="lightseagreen")
legend("bottomright", legend=c("tasa aprendizaje 0.001","tasa aprendizaje 0.01","tasa
    aprendizaje 0.1"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.001_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.1_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa aprendizaje 0.001","tasa aprendizaje 0.01","tasa
    aprendizaje 0.1"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.001_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.1_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa aprendizaje 0.001","tasa aprendizaje 0.01","tasa
    aprendizaje 0.1"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion arate 5-15-25

par(mfrow=c(1,3))

plot(memn2n_epocas100_batch32_lrate0.01_arate5_emb40_mem50_gradnorm40_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
```

```
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate25_emb40_mem50_gradnorm40_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa reducción aprendizaje 5","tasa reducción
    aprendizaje 15","tasa reducción aprendizaje 25"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate5_emb40_mem50_gradnorm40_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate25_emb40_mem50_gradnorm40_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa reducción aprendizaje 5","tasa reducción
    aprendizaje 15","tasa reducción aprendizaje 25"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate5_emb40_mem50_gradnorm40_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate25_emb40_mem50_gradnorm40_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa reducción aprendizaje 5","tasa reducción
    aprendizaje 15","tasa reducción aprendizaje 25"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion arate 0-15

par(mfrow=c(1,3))

plot(memn2n_epocas100_batch32_lrate0.01_arate0_emb40_mem50_gradnorm40_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa reducción aprendizaje 0","tasa reducción
    aprendizaje 15"), pch=15,col=c("purple","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate0_emb40_mem50_gradnorm40_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("tasa reducción aprendizaje 0","tasa reducción
    aprendizaje 15"), pch=15,col=c("purple","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate0_emb40_mem50_gradnorm40_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="lightseagreen")
```

```
legend("bottomright",legend=c("tasa reducción aprendizaje 0","tasa reducción
    aprendizaje 15"), pch=15,col=c("purple","lightseagreen"))



# comparacion maxgradnorm 20-40-80

par(mfrow=c(1,3))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm20_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm80_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("norma gradiente 20","norma gradiente 40","norma
    gradiente 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm20_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm80_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("norma gradiente 20","norma gradiente 40","norma
    gradiente 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm20_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm80_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("norma gradiente 20","norma gradiente 40","norma
    gradiente 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))



#comparacion batch 10-32-50

par(mfrow=c(1,3))

plot(memn2n_epocas100_batch10_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch50_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch10_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
```

```
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch50_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch10_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch50_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



#comparacion embedding size 20-40-80

par(mfrow=c(1,3))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb20_mem50_gradnorm40_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb80_mem50_gradnorm40_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 40","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb20_mem50_gradnorm40_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb80_mem50_gradnorm40_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 40","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb20_mem50_gradnorm40_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb80_mem50_gradnorm40_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 40","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
# comparacion memory size 20-50-80

par(mfrow=c(1,3))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem20_gradnorm40_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem80_gradnorm40_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("memoria 20","memoria 50","memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem20_gradnorm40_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem80_gradnorm40_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("memoria 20","memoria 50","memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem20_gradnorm40_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem80_gradnorm40_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("memoria 20","memoria 50","memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion hops 1-3-8 (a mas hops, mas tiempo de ejecucion)

par(mfrow=c(1,4))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto1$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto2$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 2",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto2$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto2$Test,
    col="lightseagreen")
```

```
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto3$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 3",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto3$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto3$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto4$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 4",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto4$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto4$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto5$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto6$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 6",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto6$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto6$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto7$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 7",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto7$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto7$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto8$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 8",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto8$Test,
    col="dodgerblue2")
```

```
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto8$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto9$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 9",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto9$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto9$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto10$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto11$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 11",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto11$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto11$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto12$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 12",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto12$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto12$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto13$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 13",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto13$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto13$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto14$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 14",
    type="l", col="purple")
```

```
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto14$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto14$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto15$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 15",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto15$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto15$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto16$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 16",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto16$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto16$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto17$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 17",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto17$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto17$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto18$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 18",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto18$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto18$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto19$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 19",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto19$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto19$Test,
    col="lightseagreen")
legend("topright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops1_conjunto20$Test,
```

```
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 20",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto20$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_hops8_conjunto20$Test,
    col="lightseagreen")
legend("bottomright", legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 100 epocas, maxgradnorm 40, batch 32, emb 40, mem 50, lrate 0.01, comparacion
    random-xavier unif-xavier normal

par(mfrow=c(1,3))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_xavierunif_conjunto1
   $Test, col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_xavierrandom_conjunto1
   $Test, col="lightseagreen")
legend("bottomright",legend=c("Inicialización normal", "Inicialización Xavier
    uniforme", "Inicialización Xavier normal"), pch=15, col=c("purple",
    "dodgerblue2", "lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_xavierunif_conjunto5
   $Test, col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_xavierrandom_conjunto5
   $Test, col="lightseagreen")
legend("bottomright",legend=c("Inicialización normal", "Inicialización Xavier
    uniforme", "Inicialización Xavier normal"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_xavierunif_conjunto10
   $Test, col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_xavierrandom_conjunto10
   $Test, col="lightseagreen")
legend("bottomright",legend=c("Inicialización normal", "Inicialización Xavier
    uniforme", "Inicialización Xavier normal"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 100 epocas, maxgradnorm 40, batch 32, emb 40, mem 50, lrate 0.01, comparacion
    SGD-ADAM

par(mfrow=c(1,3))
```

```
plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_ADAM_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("SGD","Adam"), pch=15,col=c("purple","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_ADAM_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("SGD","Adam"), pch=15,col=c("purple","lightseagreen"))

plot(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_ADAM_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("SGD","Adam"), pch=15,col=c("purple","lightseagreen"))
```

WikiMovies

```
# comparacion 60-100-200 epocas, en funcion de learning rate

par(mfrow=c(1,3))

plot(memn2n_peliculasfinal_hops3_epochs60_lrate0.1_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Épocas - tasa de aprendizaje 0.1", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.1_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs200_lrate0.1_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(memn2n_peliculasfinal_hops3_epochs60_lrate0.01_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Épocas - tasa de aprendizaje 0.01", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.01_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs200_lrate0.01_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(memn2n_peliculasfinal_hops3_epochs60_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Épocas - tasa de aprendizaje 0.001", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs200_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="lightseagreen")
```

```
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion hops

par(mfrow=c(1,2))

plot(memn2n_peliculasfinal_hops1_epochs100_lrate0.01_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Saltos - tasa de aprendizaje 0.01", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.01_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops8_epochs100_lrate0.01_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(memn2n_peliculasfinal_hops1_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Saltos - tasa de aprendizaje 0.001", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops8_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion learning rate 0.1-0.01-0.001 con SGD y opt adam

par(mfrow=c(1,2))

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.1_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Optimización SGD", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.01_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.1_arate200_batch10_emb40_mem50
   _gradnorm20.0_optadam$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Optimización Adam", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.01_arate200_batch10_emb40_mem50
   _gradnorm20.0_optadam$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0_optadam$Test, col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
# comparacion SGD y Adam, con arate20

par(mfrow=c(1,2))

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.1_arate20_batch10_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Optimización SGD", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.01_arate20_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate20_batch10_emb40_mem50
   _gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.1_arate20_batch10_emb40_mem50
   _gradnorm20.0_optadam$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Optimización Adam", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.01_arate20_batch10_emb40_mem50
   _gradnorm20.0_optadam$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate20_batch10_emb40_mem50
   _gradnorm20.0_optadam$Test, col="lightseagreen")
legend("bottomright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.001"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion algoritmo optimizacion SGD-ADAM

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.01_arate200_batch10_emb40_mem50
   _gradnorm20.0_optadam$Test, col="lightseagreen")
legend("bottomright",legend=c("optimización SGD","optimización Adam"),
    pch=15,col=c("purple","lightseagreen"))



# comparacion batch 5-10-30

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch5_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch30_emb40_mem50
   _gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("tamaño batch 5","tamaño batch 10","tamaño batch 30"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
# comparacion embedding 20-40-80

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb20_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb80_mem50
   _gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("tamaño inmersión 20","tamaño inmersión 40","tamaño
    inmersión 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion batch 5-10, embedding 20-40-80

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch5_emb20_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch5_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch5_emb80_mem50
   _gradnorm20.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb20_mem50
   _gradnorm20.0$Test, col="red2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="orange")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb80_mem50
   _gradnorm20.0$Test, col="gold")
legend("bottomright",legend=c("batch 5","inmersión 20","inmersión 40","inmersión
    80"), pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("batch 10","inmersión 20","inmersión 40","inmersión
    80"), pch=15,col=c("white","red2","orange","gold"))


plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb20_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb80_mem50
   _gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 40","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion inicializacion pesos normal-xavierunif-xaviernormal

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0_initxavierunif$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0_initxaviernormal$Test, col="lightseagreen")
legend("bottomright",legend=c("inicialización normal","inicialización Xavier
    uniforme","inicialización Xavier normal"),cex = 0.8,
```

```
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


# comparacion norma gradiente 10-20-40-60

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm10.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm40.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm60.0$Test, col="darkblue")
legend("bottomright",legend=c("norma gradiente 10","norma gradiente 20","norma
    gradiente 40","norma gradiente 60"), cex=0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))



# comparacion memoria 10-20-50-80

par(mfrow=c(1,2))

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem10
   _gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Con norma gradiente 20", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem20
   _gradnorm20.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm20.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem80
   _gradnorm20.0$Test, col="darkblue")
legend("bottomright",legend=c("memoria 10","memoria 20","memoria 50","memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))



# comparacion memoria 10-20-50-80, con gradnorm 40

plot(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem10
   _gradnorm40.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Con norma gradiente 40", type="l", col="purple")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem20
   _gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem50
   _gradnorm40.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_hops3_epochs100_lrate0.001_arate200_batch10_emb40_mem80
   _gradnorm40.0$Test, col="darkblue")
legend("bottomright",legend=c("memoria 10","memoria 20","memoria 50","memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))



########################################
########################################
########################################
```

```
# comparaciones formatos

plot(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Comparación formatos con MemN2N", type="l", col="purple")
lines(memn2n_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem52_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem21_gradnorm40.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem75_gradnorm40.0$Test, col="darkblue")
legend("bottomright",legend=c("frase","ventana","tripleta","frase+tripleta"),cex =
    0.8, pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))



###############
### frases ###
###############

# comparacion memoria 25-50-75-100

plot(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem25_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem50_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem75_gradnorm40.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem100_gradnorm40.0$Test, col="darkblue")
legend("bottomright",legend=c("memoria 25","memoria 50","memoria 75","memoria 100"),
    cex = 0.9, pch=15,col=c("purple","dodgerblue2","lightseagreen", "darkblue"))


plot(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem25_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem50_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem75_gradnorm40.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem100_gradnorm40.0$Test, col="darkblue")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem7_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem14_gradnorm40.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem21_gradnorm40.0$Test, col="darkblue")


# opt

plot(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
```

```
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem75_gradnorm40.0_ADAM$Test, col="dodgerblue2")

# init

plot(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem75_gradnorm40.0_XAVIER$Test, col="dodgerblue2")



#################
### ventanas ###
#################

plot(memn2n_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem100_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem150_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 50","memoria 100","memoria 150"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(OTRAVECT_memn2n_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(OTRAVECT_memn2n_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem100_gradnorm40.0$Test, col="dodgerblue2")
lines(OTRAVECT_memn2n_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem110_gradnorm40.0$Test, col="lightseagreen")
lines(OTRAVECT_memn2n_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem220_gradnorm40.0$Test, col="red2")
lines(OTRAVECT_memn2n_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem27_gradnorm40.0$Test, col="orange")
lines(OTRAVECT_memn2n_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem54_gradnorm40.0$Test, col="gold")


plot(OTRAVECT_memn2n_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(OTRAVECT_memn2n_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem54_gradnorm40.0$Test, col="gold")
lines(TODO_memn2n_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem52_gradnorm40.0$Test, col="blue")
lines(TODO_memn2n_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem78_gradnorm40.0$Test, col="darkblue")



##################
### tripletas ###
##################
```

```
plot(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem20_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem30_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 25","memoria 50","memoria 75"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem20_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem30_gradnorm40.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem7_gradnorm40.0$Test, col="red2")
lines(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem14_gradnorm40.0$Test, col="orange")
lines(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem21_gradnorm40.0$Test, col="gold")



##########################
### frases + tripletas ###
##########################

# max_story_triple_size = 10
plot(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem20_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem30_gradnorm40.0$Test, col="lightseagreen")


# max_story_size = 25
plot(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem25_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem50_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem75_gradnorm40.0$Test, col="lightseagreen")


# (max_story_size + max_story_triple_size)/2 = 17.5
plot(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem17_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem35_gradnorm40.0$Test, col="dodgerblue2")
```

```
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem52_gradnorm40.0$Test, col="lightseagreen")


# FLAGS.memory_size = 50
plot(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem100_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem150_gradnorm40.0$Test, col="lightseagreen")


# mean_story_size = 7
plot(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem7_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem14_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem21_gradnorm40.0$Test, col="lightseagreen")


# mejor entre anteriores
plot(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem21_gradnorm40.0$Test, col="dodgerblue2")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem50_gradnorm40.0$Test, col="lightseagreen")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem52_gradnorm40.0$Test, col="darkblue")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem75_gradnorm40.0$Test, col="orange")

plot(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem21_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem50_gradnorm40.0$Test, col="red")
lines(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem75_gradnorm40.0$Test, col="lightseagreen")



# comparacion formatos memn2n-kvmemnn

plot(memn2n_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Formato frase", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("MemN2N","KVMemNN"), cex = 0.9,
    pch=15,col=c("purple","lightseagreen"))
```

```
plot(memn2n_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem52_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Formato ventana", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem52_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("MemN2N","KVMemNN"), cex = 0.9,
    pch=15,col=c("purple","lightseagreen"))


plot(memn2n_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate200_batch10_emb40
   _mem21_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Formato tripleta", type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem21_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("MemN2N","KVMemNN"), cex = 0.9,
    pch=15,col=c("purple","lightseagreen"))


plot(memn2n_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate200_batch10
   _emb40_mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Formato frase + tripleta", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("MemN2N","KVMemNN"), cex = 0.9,
    pch=15,col=c("purple","lightseagreen"))
```

B.3. KVMemNN

tareas bAbI

```
# comparacion modelos
plot(rnn_LSTM_epocas200_batch32_ADAM_lrate0.1_arate0_emb40_mem50_conjunto1$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("RNN","KVMemNN","MemN2N"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(rnn_LSTM_epocas200_batch32_ADAM_lrate0.1_arate0_emb40_mem50_conjunto5$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("RNN","KVMemNN","MemN2N"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(rnn_LSTM_epocas200_batch32_ADAM_lrate0.1_arate0_emb40_mem50_conjunto10$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
```

```
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(memn2n_epocas100_batch32_lrate0.01_arate15_emb40_mem50_gradnorm40_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("RNN","KVMemNN","MemN2N"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion 60-100-200 epocas (test), con lrate 0.01,batch 32 y gradnorm 40

par(mfrow=c(1,3))

plot(kvmemnn_epocas60_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(kvmemnn_epocas100_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas60_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(kvmemnn_epocas100_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas60_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(kvmemnn_epocas100_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


# comparacion 60-100-200 epocas (test), con lrate 0.001, batch 50 y gradnorm 20

par(mfrow=c(1,3))

plot(kvmemnn_epocas60_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 1",
    type="l", col="purple")
lines(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="red2")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
```

```
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas60_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 5",
    type="l", col="purple")
lines(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="red2")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas60_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea 10",
    type="l", col="purple")
lines(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="red2")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# train, valid y test

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto1$Train,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto1$Valid,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("Train","Valid", "Test"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto5$Train,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto5$Valid,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("Train","Valid", "Test"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto10$Train,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto10$Valid,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
```

```
    col="lightseagreen")
legend("bottomright",legend=c("Train","Valid", "Test"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, comparacion maxgradnorm 20-40

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("maxgradnorm40","maxgradnorm20"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("maxgradnorm40","maxgradnorm20"),
    pch=15,col=c("purple", "lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("maxgradnorm40","maxgradnorm20"),
    pch=15,col=c("purple","lightseagreen"))



# 200 epocas, comparacion maxgradnorm 20-40-80

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm80_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("norma gradiente 20","norma gradiente 40","norma
    gradiente 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm80_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("norma gradiente 20","norma gradiente 40","norma
```

```
    gradiente 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm80_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("norma gradiente 20","norma gradiente 40","norma
    gradiente 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, lrate 0.01, maxgradnorm 20, comparacion batch 32-50

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch32","batch50"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch32","batch50"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch32","batch50"),
    pch=15,col=c("purple","lightseagreen"))



# 200 epocas, lrate 0.001, maxgradnorm 20-40, comparacion batch 32-50

par(mfrow=c(2,3))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="red2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="gold")
legend("bottomright",legend=c("gradnorm20","batch32","batch50"),
```

```
    pch=15,col=c("white","purple","lightseagreen"))
legend("bottomleft",legend=c("gradnorm40","batch32","batch50"),
    pch=15,col=c("white","red2","gold"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="red2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="gold")
legend("bottomright",legend=c("gradnorm20","batch32","batch50"),
    pch=15,col=c("white","purple","lightseagreen"))
legend("bottomleft",legend=c("gradnorm40","batch32","batch50"),
    pch=15,col=c("white","red2","gold"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="red2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="gold")
legend("bottomright",legend=c("gradnorm20","batch32","batch50"),
    pch=15,col=c("white","purple","lightseagreen"))
legend("bottomleft",legend=c("gradnorm40","batch32","batch50"),
    pch=15,col=c("white","red2","gold"))



# 200 epocas, lrate 0.001, maxgradnorm 40, comparacion batch 10-32-50

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch10_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch10_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch 32","batch 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
plot(kvmemnn_epocas200_batch10_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("batch 10","batch32","batch50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, lrate 0.001, maxgradnorm 40, batch 32, comparacion hops 1-3-8

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_hops1_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_hops8_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_hops1_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_hops8_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_hops1_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_hops8_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, lrate 0.001, maxgradnorm 40, batch 32, hops 3, feature 50, comparacion
    embedding 20-40-80

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb20_mem50_gradnorm40_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="dodgerblue2")
```

```
lines(kvmemnn_epocas200_batch32_lrate0.001_emb80_mem50_gradnorm40_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 40","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb20_mem50_gradnorm40_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb80_mem50_gradnorm40_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 40","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb20_mem50_gradnorm40_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb80_mem50_gradnorm40_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("inmersión 20","inmersión 40","inmersión 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="dodgerblue2")



# 200 epocas, lrate 0.001, maxgradnorm 40, batch 32, hops 3, embedding 40,
    comparacion feature 10-20-50

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat10_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat20_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("características 10","características
    20","características 50"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat10_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat20_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("características 10","características
    20","características 50"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat10_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat20_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("características 10","características
    20","características 50"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, lrate 0.001, feat 20-50, comparacion embedding 20-40-80

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb20_mem50_gradnorm40_feat20_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat20_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb80_mem50_gradnorm40_feat20_conjunto1$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb20_mem50_gradnorm40_feat50_conjunto1$Test,
    col="red2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="orange")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb80_mem50_gradnorm40_feat50_conjunto1$Test,
    col="gold")
legend("bottomright",legend=c("características 20","inmersión 20","inmersión 40",
    "inmersión 80"), pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("características 50","inmersión 20","inmersión 40",
    "inmersión 80"), pch=15,col=c("white","red2","orange","gold"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb20_mem50_gradnorm40_feat20_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat20_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb80_mem50_gradnorm40_feat20_conjunto5$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb20_mem50_gradnorm40_feat50_conjunto5$Test,
    col="red2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="orange")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb80_mem50_gradnorm40_feat50_conjunto5$Test,
    col="gold")
legend("bottomright",legend=c("características 20","inmersión 20","inmersión 40",
    "inmersión 80"), pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("características 50","inmersión 20","inmersión 40",
    "inmersión 80"), pch=15,col=c("white","red2","orange","gold"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb20_mem50_gradnorm40_feat20_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat20_conjunto10$Test,
    col="dodgerblue2")
```

```
lines(kvmemnn_epocas200_batch32_lrate0.001_emb80_mem50_gradnorm40_feat20_conjunto10$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb20_mem50_gradnorm40_feat50_conjunto10$Test,
    col="red2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="orange")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb80_mem50_gradnorm40_feat50_conjunto10$Test,
    col="gold")
legend("bottomright",legend=c("características 20","inmersión 20","inmersión 40",
    "inmersión 80"), pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("características 50","inmersión 20","inmersión 40",
    "inmersión 80"), pch=15,col=c("white","red2","orange","gold"))



# 200 epocas, lrate 0.001, feat 50, embedding 40, batch 32-50, comparacion memory
    20-50-80

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem20_gradnorm40_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem80_gradnorm40_feat50_conjunto1$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem20_gradnorm40_feat50_conjunto1$Test,
    col="red2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="orange")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem80_gradnorm40_feat50_conjunto1$Test,
    col="gold")
legend("bottomright",legend=c("batch 32","memoria 20","memoria 50", "memoria 80"),
    pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("batch 50","memoria 20","memoria 50", "memoria 80"),
    pch=15,col=c("white","red2","orange","gold"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem20_gradnorm40_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem80_gradnorm40_feat50_conjunto5$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem20_gradnorm40_feat50_conjunto5$Test,
    col="red2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="orange")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem80_gradnorm40_feat50_conjunto5$Test,
    col="gold")
legend("bottomright",legend=c("batch 32","memoria 20","memoria 50", "memoria 80"),
    pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("batch 50","memoria 20","memoria 50", "memoria 80"),
    pch=15,col=c("white","red2","orange","gold"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem20_gradnorm40_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
```

```
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem80_gradnorm40_feat50_conjunto10$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem20_gradnorm40_feat50_conjunto10$Test,
    col="red2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="orange")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem80_gradnorm40_feat50_conjunto10$Test,
    col="gold")
legend("bottomright",legend=c("batch 32","memoria 20","memoria 50", "memoria 80"),
    pch=15,col=c("white","purple","dodgerblue2","lightseagreen"))
legend("bottomleft",legend=c("batch 50","memoria 20","memoria 50", "memoria 80"),
    pch=15,col=c("white","red2","orange","gold"))



par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem20_gradnorm40_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem80_gradnorm40_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("memoria 20","memoria 50", "memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem20_gradnorm40_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem80_gradnorm40_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("memoria 20","memoria 50", "memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem20_gradnorm40_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem80_gradnorm40_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("memoria 20","memoria 50", "memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto2$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    2", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto2$Test,
    col="lightseagreen")
```

```
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto2$Test,
    col="red2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto2$Test,
    col="gold")
legend("bottomright",legend=c("gradnorm20","batch32","batch50"),
    pch=15,col=c("white","purple","lightseagreen"))
legend("bottomleft",legend=c("gradnorm40","batch32","batch50"),
    pch=15,col=c("white","red2","gold"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto6$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    6", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto6$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto6$Test,
    col="red2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto6$Test,
    col="gold")
legend("bottomright",legend=c("gradnorm20","batch32","batch50"),
    pch=15,col=c("white","purple","lightseagreen"))
legend("bottomleft",legend=c("gradnorm40","batch32","batch50"),
    pch=15,col=c("white","red2","gold"))

plot(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto11$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    11", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto11$Test,
    col="lightseagreen")
lines(kvmemnn_epocas200_batch32_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto11$Test,
    col="red2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm40_feat50_conjunto11$Test,
    col="gold")
legend("bottomright",legend=c("gradnorm20","batch32","batch50"),
    pch=15,col=c("white","purple","lightseagreen"))
legend("bottomleft",legend=c("gradnorm40","batch32","batch50"),
    pch=15,col=c("white","red2","gold"))



# 200 epocas, maxgradnorm 20, batch 50, comparacion lrate 0.10.01-0.001

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.1_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
legend("topleft",legend=c("tasa de aprendizaje 0.001","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.1"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
```

```
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.1_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
legend("topleft",legend=c("tasa de aprendizaje 0.001","tasa de aprendizaje
    0.01","tasa de aprendizaje 0.1"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.1_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
legend("topleft",legend=c("lrate 0.001","lrate 0.01","lrate 0.1"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, maxgradnorm 20, batch 50, lrate 0.001, comparacion emb 20-40-80

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb20_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb80_mem50_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("embedding 20","embedding 40", "embedding 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb20_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb80_mem50_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("embedding 20","embedding 40", "embedding 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb20_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb80_mem50_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("embedding 20","embedding 40", "embedding 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, maxgradnorm 40, batch 50, lrate 0.001, comparacion emb 20-40-80
# 200 epocas, maxgradnorm 20, batch 50, lrate 0.001, emb 40, comparacion mem 20-50-80
```

```
par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb20_mem20_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem40_gradnorm20_feat50_conjunto1$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb80_mem80_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("memoria 20","memoria 40","memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb20_mem20_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem40_gradnorm20_feat50_conjunto5$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb80_mem80_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("memoria 20","memoria 40","memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb20_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem20_gradnorm20_feat50_conjunto10$Test,
    col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb80_mem20_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("memoria 20","memoria 40","memoria 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, maxgradnorm 20, batch 50, lrate 0.001, emb 40, mem 50, comparacion feat
    50-20

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat20_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("feature 50","feature 20"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat20_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("feature 50","feature 20"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
```

```
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat20_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("feature 50","feature 20"),
    pch=15,col=c("purple","lightseagreen"))



# 100 epocas, maxgradnorm 20, batch 32-50, emb 40, mem 50, comparacion lrate
    0.01-0.001

par(mfrow=c(1,3))

plot(kvmemnn_epocas100_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto1$Test,
    xlim = c(0,100), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("lrate 0.01","lrate 0.001"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas100_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto5$Test,
    xlim = c(0,100), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("lrate 0.01","lrate 0.001"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas100_batch32_lrate0.01_emb40_mem50_gradnorm40_feat50_conjunto10$Test,
    xlim = c(0,100), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("lrate 0.01","lrate 0.001"),
    pch=15,col=c("purple","lightseagreen"))



# 100 epocas, maxgradnorm 20, batch 50, emb 40, mem 50, comparacion lrate 0.01-0.001

par(mfrow=c(1,3))

plot(kvmemnn_epocas100_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,100), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("lrate 0.01","lrate 0.001"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas100_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,100), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("lrate 0.01","lrate 0.001"),
    pch=15,col=c("purple","lightseagreen"))
```

```
plot(kvmemnn_epocas100_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,100), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("lrate 0.01","lrate 0.001"),
    pch=15,col=c("purple","lightseagreen"))



# 100-200 epocas, maxgradnorm 20, batch 50, emb 40, mem 50, lrate 0.001

par(mfrow=c(1,3))

plot(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("100 epocas","200 epocas"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("100 epocas","200 epocas"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas100_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("100 epocas","200 epocas"),
    pch=15,col=c("purple","lightseagreen"))



# 200 epocas, maxgradnorm 20, batch 50, emb 40, mem 50, feature 05, lrate 0.01,
    comparacion ADAM-SGD

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_SGD_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("ADAM","SGD"), pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_SGD_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("ADAM","SGD"), pch=15,col=c("purple","lightseagreen"))
```

```
plot(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.01_emb40_mem50_gradnorm20_feat50_SGD_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("ADAM","SGD"), pch=15,col=c("purple","lightseagreen"))



# 200 epocas, maxgradnorm 20, batch 50, emb 40, mem 50, feature 05, lrate 0.001,
    comparacion ADAM-SGD

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_SGD_conjunto1$Test,
    col="lightseagreen")
legend("bottomright",legend=c("ADAM","SGD"), pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_SGD_conjunto5$Test,
    col="lightseagreen")
legend("bottomright",legend=c("ADAM","SGD"), pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_SGD_conjunto10$Test,
    col="lightseagreen")
legend("bottomright",legend=c("ADAM","SGD"), pch=15,col=c("purple","lightseagreen"))



# comparacion ADAM con eps 0.1 - 0.001 - 0.0000001

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50
   _conjunto1$Test, xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy",
       main = "Tarea 1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_eps0.001
   _conjunto1$Test, col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_eps0.0000001
   _conjunto1$Test, col="lightseagreen")
legend("bottomright",legend=c("ADAM con epsilon 0.1","ADAM con epsilon 0.001", "ADAM
    con epsilon 0.0000001"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_eps0.001
   _conjunto5$Test, col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_eps0.0000001
```

```
   _conjunto5$Test, col="lightseagreen")
legend("bottomright",legend=c("ADAM con epsilon 0.1","ADAM con epsilon 0.001", "ADAM
    con epsilon 0.0000001"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_eps0.001
   _conjunto10$Test, col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_eps0.0000001
   _conjunto10$Test, col="lightseagreen")
legend("bottomright",legend=c("ADAM con epsilon 0.1","ADAM con epsilon 0.001", "ADAM
    con epsilon 0.0000001"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# 200 epocas, maxgradnorm 20, batch 50, emb 40, mem 50, feature 05, lrate 0.001,
    comparacion xavier-random

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_random_conjunto1
   $Test, col="lightseagreen")
legend("bottomright",legend=c("Inicializador Xavier Unif","Normal random"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_random_conjunto5
   $Test, col="lightseagreen")
legend("bottomright",legend=c("Inicializador Xavier Unif","Normal random"),
    pch=15,col=c("purple","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_random_conjunto10
   $Test, col="lightseagreen")
legend("bottomright",legend=c("Inicializador Xavier Unif","Normal random"),
    pch=15,col=c("purple","lightseagreen"))



# 200 epocas, maxgradnorm 20, batch 50, emb 40, mem 50, feature 05, lrate 0.001,
    comparacion xavier unif-xavier normal-random

par(mfrow=c(1,3))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto1$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    1", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50
   _xaviernormal_conjunto1$Test, col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_random_conjunto1
```

```
   $Test, col="lightseagreen")
legend("bottomright",legend=c("Inicialización Xavier uniforme","Inicialización Xavier
    normal","Inicialización normal"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto5$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    5", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_xaviernormal
   _conjunto5$Test, col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_random_conjunto5
   $Test, col="lightseagreen")
legend("bottomright",legend=c("Inicialización Xavier uniforme","Inicialización Xavier
    normal","Inicialización normal"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_conjunto10$Test,
    xlim = c(0,200), ylim = c(0,1), xlab="Épocas", ylab= "Accuracy", main = "Tarea
    10", type="l", col="purple")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_xaviernormal
   _conjunto10$Test, col="dodgerblue2")
lines(kvmemnn_epocas200_batch50_lrate0.001_emb40_mem50_gradnorm20_feat50_random_conjunto10
   $Test, col="lightseagreen")
legend("bottomright",legend=c("Inicialización Xavier uniforme","Inicialización Xavier
    normal","Inicialización normal"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

WikiMovies

```
# comparacion hops 1-3-8, en funcion de learning rate

par(mfrow=c(1,2))

plot(kvmemnn_peliculasfinal_hops1_epochs100_lrate0.01_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Saltos - tasa de aprendizaje 0.01", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.01_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops8_epochs100_lrate0.01_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_peliculasfinal_hops1_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Saltos - tasa de aprendizaje 0.001", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops8_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("1 salto","3 saltos","8 saltos"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
# comparacion 60-100-200 epocas, en funcion de learning rate

par(mfrow=c(1,3))

plot(kvmemnn_peliculasfinal_hops3_epochs60_lrate0.1_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Épocas - tasa de aprendizaje 0.1", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.1_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs200_lrate0.1_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_hops3_epochs60_lrate0.01_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Épocas - tasa de aprendizaje 0.01", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.01_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_hops3_epochs60_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "Épocas - tasa de aprendizaje 0.001", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs200_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion learning rate 0.1-0.01-0.001

plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.1_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.01_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs200_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("tasa aprendizaje 0.1","tasa aprendizaje 0.01","tasa
    aprendizaje 0.001"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion batch 5-10-15-30

plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch5_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
```

```
      "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch15_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="darkblue")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch30_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("batch 5","batch 10","batch 15","batch 30"),
    pch=15,col=c("purple","dodgerblue2","darkblue","lightseagreen"))



# comparacion tamaño feature 10-20-40-80

plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat10
   _emb40_mem50_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
      "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="darkblue")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat80
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("tamaño características 10","tamaño características
    20","tamaño características 80"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion tamaño inmersion 20-40-80, con feature 20 y 40

par(mfrow=c(1,2))

plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb20_mem50_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
      "Accuracy", main = "Inmersión - características tamaño 20", type="l",
      col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb80_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("tamaño inmersión 20","tamaño inmersión 40","tamaño
    inmersión 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))

plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb20_mem50_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
      "Accuracy", main = "Inmersión - características tamaño 40", type="l",
      col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat40
   _emb80_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("tamaño inmersión 20","tamaño inmersión 40","tamaño
    inmersión 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion tamaño inmersion 20-40-80
```

```
plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb20_mem50_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb80_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("tamaño inmersión 20","tamaño inmersión 40","tamaño
    inmersión 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion tamaño norma gradiente 10-20-40

plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm10.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("tamaño norma gradiente 10","tamaño norma gradiente
    20","tamaño norma gradiente 40"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion tamaño memoria 10-20-40-80

plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem10_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem20_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem80_gradnorm20.0$Test, col="darkblue")
legend("bottomright",legend=c("tamaño memoria 10","tamaño memoria 20","tamaño memoria
    50", "tamaño memoria 80"), pch=15,col=c("purple","dodgerblue2","lightseagreen",
    "darkblue"))



# comparacion inicializacion normal - xavier uniforme - xavier normal

plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0_initnormal$Test, xlim=c(0,100), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0_initxavierunif$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("bottomright",legend=c("Inicialización normal","Inicialización Xavier
    uniforme", "Inicialización Xavier normal"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))
```

```
# comparacion optimizacion Adam - SGD

plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm20.0_optSGD$Test, col="lightseagreen")
legend("bottomright",legend=c("Adam","SGD"), pch=15,col=c("purple","lightseagreen"))



plot(kvmemnn_peliculasfinal_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1_feat20
   _emb40_mem50_gradnorm40.0$Test, xlim=c(0,100), ylim=c(0,1), xlab="Épocas", ylab=
       "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_sentenceMEMMAX_hops3_epochs100_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_sentenceMEM3_hops3_epochs100_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, col="lightseagreen")



#######################################################

##### tripletas


# comparacion epocas - tasa de aprendizaje

par(mfrow=c(1,3))

plot(kvmemnn_peliculasfinal_triple_hops3_epochs60_lrate0.1_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.1", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs100_lrate0.1_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.1_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_triple_hops3_epochs60_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.01", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs100_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_triple_hops3_epochs60_lrate0.001_arate10000_batch10_eps0.1
```

```
   _feat20_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.001", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.001_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



plot(kvmemnn_peliculasfinal_triple_hops3_epochs60_lrate0.1_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.1", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs100_lrate0.1_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.1_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_triple_hops3_epochs60_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.01", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs100_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_triple_hops3_epochs60_lrate0.001_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.001", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs100_lrate0.001_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.001_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion epocas - feature

plot(kvmemnn_peliculasfinal_triple_hops3_epochs60_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tamaño características 20", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs100_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
```

```
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1_feat20_emb40
    col="lightseagreen")
legend("topright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_triple_hops3_epochs60_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tamaño características 10", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs100_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1_feat10_emb40
    col="lightseagreen")
legend("topright",legend=c("60 épocas","100 épocas","200 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion batch 5-10-20

plot(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch5_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.01", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.001_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch20_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("batch 5","batch 10","batch 20"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion embedding 10-20-40-80

plot(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb10_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.01", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb20_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb80_mem50_gradnorm20.0$Test, col="darkblue")
legend("topright",legend=c("inmersión 10","inmersión 20","inmersión 40","inmersión
    80"), pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))


plot(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb10_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.01", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb20_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
```

```
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb80_mem50_gradnorm20.0$Test, col="darkblue")
legend("topright",legend=c("inmersión 10","inmersión 20","inmersión 40","inmersión
    80"), pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))



# comparacion memoria 10-20-50

plot(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem10_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.01", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem20_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
legend("topright",legend=c("memoria 10","memoria 20","memoria 50"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion feature 5-10-20-40

plot(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat5_emb40_mem50_gradnorm20.0$Test, xlim=c(0,200), ylim=c(0,1), xlab="Épocas",
       ylab= "Accuracy", main = "Épocas - tasa de aprendizaje 0.01", type="l",
       col="purple")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat20_emb40_mem50_gradnorm20.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_triple_hops3_epochs200_lrate0.01_arate10000_batch10_eps0.1
   _feat10_emb40_mem50_gradnorm20.0$Test, col="darkblue")
legend("topright",legend=c("característica 5","característica 10","característsitca
    20","característsitca 40"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))



########################################
########################################
########################################


# comparacion formatos
plot(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "Comparación formatos con KVMemNN",
       type="l", col="purple")
lines(TODO_kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem52_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem21_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="darkblue")
legend("bottomright",legend=c("frase","ventana","tripleta","frase+tripleta"),cex =
    0.8, pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))
```

```
###############
### frases ###
###############

# comparacion epocas 50-100-200

par(mfrow=c(1,2))

plot(kvmemnn_peliculasfinal_levelsentence_hops3_epochs50_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem25_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "Épocas - tamaño memoria 25",
       type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs100_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem25_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem25_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("50 épocas","100 épocas","200 épocas"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_levelsentence_hops3_epochs50_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "Épocas - tamaño memoria 75",
       type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs100_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("50 épocas","100 épocas","200 épocas"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion memoria 25-50-75

plot(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem25_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 25","memoria 50","memoria 75"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem25_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem7_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
```

```
   _eps0.1_feat20_emb40_mem14_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem21_gradnorm40.0$Test, col="dodgerblue2")



# comparacion feat 20-40

plot(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "Comparación memoria -
       características", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat40_emb40_mem50_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat40_emb40_mem75_gradnorm40.0$Test, col="darkblue")
legend("bottomright",legend=c("mem50-feat20","mem50-feat40","mem75-feat20","mem75-feat40"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))



# comparacion saltos 3-5

plot(kvmemnn_peliculasfinal_levelsentence_hops3_epochs100_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "Comparación memoria -
       características", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentence_hops5_epochs100_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_levelsentence_hops5_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="darkblue")
legend("bottomright",legend=c("ep100-hop3","ep100-hop5","ep200-hop3","ep200-hop5"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))


plot(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "Comparación memoria -
       características", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentence_hops5_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("ep100-hop3","ep100-hop5","ep200-hop3","ep200-hop5"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))



#################
### ventanas ###
#################

# comparacion memoria 50 - 100 - 150

plot(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
```

```
   _eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem100_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem150_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 50","memoria 100","memoria 150"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem150_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat40_emb40_mem100_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 50","memoria 150","memoria 100"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem100_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem150_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_levelwindow5_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem100_gradnorm40.0$Test, col="orange")
lines(kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem100_gradnorm40.0$Test, col="gold")
lines(kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem220_gradnorm40.0$Test, col="red")
lines(kvmemnn_peliculasfinal_levelwindow5_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem220_gradnorm40.0$Test, col="blue")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem220_gradnorm40.0$Test, col="darkblue")

lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, col="darkblue")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem100_gradnorm40.0$Test, col="darkblue")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem110_gradnorm40.0$Test, col="darkblue")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem220_gradnorm40.0$Test, col="darkblue")


plot(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem100_gradnorm40.0$Test, col="dodgerblue2")
#lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem110_gradnorm40.0$Test, col="lightseagreen")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem220_gradnorm40.0$Test, col="lightseagreen")
```

```
#lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem330_gradnorm40.0$Test, col="darkblue")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem27_gradnorm40.0$Test, col="blue")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem54_gradnorm40.0$Test, col="red")


plot(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem54_gradnorm40.0$Test, col="dodgerblue2")
lines(TODO_kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem52_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem52_gradnorm40.0$Test, col="darkblue")
lines(kvmemnn_peliculasfinal_levelwindow_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem78_gradnorm40.0$Test, col="darkgreen")




##################
### tripletas ###
##################

# comparacion memoria 5-10-20-30 (triple_size/2, triple_size, 2*triple_size,
    3*triple_size)

plot(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem5_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem20_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem30_gradnorm40.0$Test, col="darkblue")
legend("topright",legend=c("memoria 5","memoria 10","memoria 20","memoria 30"),cex =
    0.9, pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))


plot(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem5_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem20_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem30_gradnorm40.0$Test, col="darkblue")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem7_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem14_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem21_gradnorm40.0$Test, col="darkblue")
```

```
# comparacion feat 10-20-40

plot(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat10_emb40_mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat40_emb40_mem10_gradnorm40.0$Test, col="lightseagreen")
legend("topright",legend=c("características 10","características 20","características
    40"), cex = 0.9, pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion epocas 100-200-300

plot(kvmemnn_peliculasfinal_leveltriple_hops3_epochs100_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem20_gradnorm40.0$Test, xlim=c(0,300), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem20_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs300_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem20_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("100 épocas","200 épocas","300 épocas"),
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


par(mfrow=c(1,2))

plot(kvmemnn_peliculasfinal_leveltriple_hops3_epochs100_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem20_gradnorm40.0$Test, xlim=c(0,300), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "Épocas - tamaño de memoria 20",
       type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem20_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs300_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem20_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("100 épocas","200 épocas","300 épocas"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


plot(kvmemnn_peliculasfinal_leveltriple_hops3_epochs100_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, xlim=c(0,300), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "Épocas - tamaño de memoria 10",
       type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs300_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("200 épocas","300 épocas"), cex = 0.9,
    pch=15,col=c("purple","lightseagreen"))



# comparacion embedding 20-40-80
```

```
plot(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb20_mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb80_mem10_gradnorm40.0$Test, col="lightseagreen")
legend("topright",legend=c("inmersión 20","inmersión 40", "inmersión 80"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion batch 5-10-20

plot(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch5
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch20
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="lightseagreen")
legend("topright",legend=c("batch 5","batch 10","batch 20"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



# comparacion learning rate 0.1-0.01-0.001

plot(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.1_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.01_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="lightseagreen")
legend("topright",legend=c("tasa de aprendizaje 0.1","tasa de aprendizaje 0.01","tasa
    de aprendizaje 0.001"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))



#########################
### frases+tripletas ###
#########################

# comparacion memorias 30-50-75  (30 = 3*triple_story_size, 50 = 2*story_size, 75 =
    3*story_size)

plot(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem30_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 30","memoria 50","memoria 75"), cex = 0.9,
```

```
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


#max_story_size = 25

plot(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem25_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 30","memoria 50","memoria 75"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


#max_story_triple_size = 10

plot(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem20_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem30_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 30","memoria 50","memoria 75"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


#FLAGS.memory_size = 50

plot(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem100_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem150_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 30","memoria 50","memoria 75"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


#mean_story_size = 7

plot(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem7_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem14_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem21_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 30","memoria 50","memoria 75"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


#(max_story_size + max_story_triple_size)/2 = 17

plot(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
```

```
   _batch10_eps0.1_feat20_emb40_mem17_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem35_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem52_gradnorm40.0$Test, col="lightseagreen")
legend("bottomright",legend=c("memoria 30","memoria 50","memoria 75"), cex = 0.9,
    pch=15,col=c("purple","dodgerblue2","lightseagreen"))


#mejor de anteriores

plot(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem14_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem50_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem52_gradnorm40.0$Test, col="darkblue")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="darkgreen")



### optimos

plot(kvmemnn_peliculasfinal_levelsentence_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, xlim=c(0,200), ylim=c(0,1),
       xlab="Épocas", ylab= "Accuracy", main = "", type="l", col="purple")
lines(OTRAVECT_kvmemnn_peliculasfinal_levelwindow7_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem54_gradnorm40.0$Test, col="dodgerblue2")
lines(kvmemnn_peliculasfinal_leveltriple_hops3_epochs200_lrate0.001_arate10000_batch10
   _eps0.1_feat20_emb40_mem10_gradnorm40.0$Test, col="lightseagreen")
lines(kvmemnn_peliculasfinal_levelsentencetriple_hops3_epochs200_lrate0.001_arate10000
   _batch10_eps0.1_feat20_emb40_mem75_gradnorm40.0$Test, col="darkblue")
legend("topright",legend=c("frase","ventana","tripleta","frase+tripleta"),cex = 0.8,
    pch=15,col=c("purple","dodgerblue2","lightseagreen","darkblue"))
```