Escuela Técnica Superior de Ingenieros Informáticos

Universidad Politécnica de Madrid

Razonamiento Automático sobre Bases de Conocimiento mediante Deep Learning

Trabajo Fin de Máster

Máster en Inteligencia Artificial

AUTOR: Elvira Amador Domínguez
TUTORES: Emilio Serrano Fernández
y Daniel Manrique Gamo

2017/2018

Agradecimientos

Mis agradecimientos a Francisco Laport López, egresado del Máster Universitario en Inteligencia Artificial por la elaboración del código LaTeX empleado en la redacción de esta memoria.

Al profesor Thomas Lukasiewicz del "Department of Computer Science" de la Universidad de Oxford por su importante contribución a la hipótesis científica presentada en este trabajo, y por dar acceso al servidor de su grupo de investigación que permitió realizar los experimentos que apoyan esta hipótesis.

A mis compañeros de laboratorio, por su ayuda desinteresada en todo momento, por permitirme aprender de ellos cada día y por ser las grandes personas que son.

A Emilio y a Daniel, por confiar en mí y compartir conmigo su tiempo y su conocimiento.

A mi familia, por apoyarme en cada una de mis decisiones y acompañarme a lo largo del camino.

A Tamara, por su infinita paciencia conmigo y por tener siempre palabras de ánimo para darme.

A Gonzalo, por ser mi gran apoyo y por creer en mí incluso cuando a mí me costaba hacerlo.

A mi padre, porque nunca ha dejado de estar conmigo, y nunca dejo ni dejaré de aprender de él.

Resumen

La representación del conocimiento es una de las principales áreas de estudio dentro de la Inteligencia Artificial. Una forma de modelización son las bases de conocimiento (*Knowledge Bases* o KB), tales como *WordNet* o *Freebase*, que dio lugar a la creación del *Google Knowledge Graph*. Estos modelos de conocimiento permiten llevar a cabo un gran número de tareas, entre las cuales, una de las más interesantes es la posibilidad de devolver respuestas inteligentes y personalizadas a preguntas específicas, gracias al razonamiento semántico realizado sobre las mismas. Sin embargo, estos modelos presentan un gran problema: están incompletos.

Para solucionar este problema, ha habido un interés creciente en el *Statistical Relational Learning* (SRL), o Aprendizaje Estadístico Relacional, el cual permite, por ejemplo, emplear técnicas de aprendizaje automático para llevar a cabo razonamiento sobre datos relacionales. En este trabajo, se propone un método que combina técnicas de este tipo con una de las formas de representación del conocimiento más tradicionales de la Inteligencia Artificial: las ontologías. Con la introducción de información ontológica explícita, se espera que el modelo de razonamiento no sólo mejore su capacidad predictiva sino que, incluso, sea capaz de llevar a cabo inferencias sobre datos no existentes previamente en la base de conocimiento.

De acuerdo con el método propuesto, una entidad puede ser contextualizada por medio de la jerarquía de clases que la representa en una ontología, lo que, indirectamente, permite al modelo predictivo inferir restricciones acerca de las relaciones basadas en esta información, determinando así qué tipos de entidades componen un hecho en una relación dada. En este trabajo, se plantea el uso de este método aplicado al problema específico de predicción de tripletas, es decir, dado un hecho compuesto por dos entidades y la relación que las une, se busca determinar si éste es es, o no, factible.

Para probar esta hipótesis y, tras realizar una revisión de los trabajos científicos en la literatura especializada, se selecciona un modelo ya existente de razonamiento sobre bases de conocimiento: la red de neuronas tensorial. En primer lugar, se realizan una serie de experimentos destinados a comprobar la capacidad predictiva del modelo para, a continuación, llevar a cabo una experimentación pormenorizada en la que se combina este modelo con el método de introducción ontológica propuesto. Tras la experimentación, se comprueba que la inclusión de este tipo de conocimiento a la hora de llevar a cabo las predicciones reporta una mejora sensible en los resultados, permitiendo, además, razonar sobre hechos en los que se incluyen entidades nuevas no observadas en fase de entrenamiento.

Abstract

Knowledge representation is one of the main areas of focus inside Artificial Intelligence. Knowledge Bases are a particular way of knowledge modelling, and *WordNet* and *Freebase* are examples of it. This last one served as base for the generation of the *Google Knowledge Graph*. These models are able to perform a large number of tasks, and one of the most relevant ones is the possibility of obtaining personalized and intelligent answers for specific questions through intelligent reasoning. However, these models have a great downside: they are incomplete.

To solve these existing difficulties, there has been a growing interest in Statistical Relational Data (SRL), which allows, for example, the use of machine learning techniques to reason over relational data. In this work, we propose a method that combines these recent techniques with one of the most tradicional knowledge representation models in Artificial Intelligence: ontologies. With the introduction of explicit ontological information, we expect the model not only to improve its predictive capability, but also to even be able to perform reasoning over data previously non-existent on the knowledge base.

According to the method proposed in this work, an entity can be contextualized by means of the class hierarchy that represents the entity in an ontology, what indirectly allows the model to infer restrictions about the relations based on this information, determining which types of entities form a fact for a certain relation. This work is focused on the application of this method for the specific problem of triple prediction in which, given a fact composed by two entities and the relation that joins them, the goal is to predict whether this fact is feasible or not.

To prove this hypothesis and, after performing a highly detailed study of the avalible methods, we decide to use the neural tensor network as the reasoning model. First of all, we replicate the results reported for the neural tensor network in order to prove the reasoning capability of the model and, after that, we conduct a series of experiments where we combine the original model with the proposed method of introduction of ontological information. After this experimentation, the results show that the inclusion of this kind of knowledge on the predictive model induces a little improvement on the results, as well as enabling reasoning over facts that include new entities that have not been previously seen by the model during the training phase.

Índice

1. INTRODUCCIÓN . . . 3
1.1. Introducción . . . 3
1.2. Motivación . . . 4
1.3. Objetivos . . . 4
1.4. Estructura del documento . . . 5
2. REDES DE NEURONAS ARTIFICIALES . . . 7
2.1. Aprendizaje . . . 8
2.2. Redes de neuronas alimentadas hacia delante, o *feed-forward* . . . 10
2.3. Redes de neuronas recurrentes . . . 10
2.4. Evolución de las redes de neuronas . . . 11
2.5. *Deep Learning* . . . 13
3. REPRESENTACIÓN DISTRIBUIDA DEL LENGUAJE: *WORD EMBEDDING* . . . 19
3.1. Origen . . . 19
3.2. Modelos basados en conteo . . . 20
3.3. Modelos basados en predicción . . . 21
3.4. Modelos basados en tareas . . . 23
3.5. Otras aproximaciones . . . 24
4. BASES DE CONOCIMIENTO . . . 25
4.1. Bases de conocimiento empleadas habitualmente . . . 26
4.2. Ontologías . . . 28
4.3. Compleción de Bases de Conocimiento . . . 29
4.3.1. Métodos principales de *Knowledge Base Completion* . . . 31
4.3.2. Otras aproximaciones . . . 36
4.4. Limitaciones y problemas encontrados . . . 37
5. PLANTEAMIENTO DEL PROBLEMA . . . 39
5.1. Dificultades encontradas . . . 40
6. SOLUCIÓN PROPUESTA . . . 41
6.1. Planteamiento general de la solución . . . 41
6.2. Consideraciones previas . . . 42
6.2.1. Selección del modelo predictivo . . . 42
6.2.2. Introducción de información ontológica . . . 42
6.2.3. Bases de conocimiento empleadas . . . 43
6.2.4. Fuentes de conocimiento ontológico empleadas . . . 45
6.3. Fase 1: Implementación del modelo predictivo . . . 45
6.3.1. Diseño e inicialización . . . 46
6.3.2. Especificación del conjunto de operaciones realizadas . . . 47
6.3.3. Función de pérdida . . . 49
6.3.4. Método de evaluación . . . 50
6.4. Fase 2: Replicación de los resultados originales . . . 51
6.4.1. Replicación resultados en WordNet . . . 52
6.4.2. Replicación resultados en Freebase . . . 52

6.4.3. Discusión de resultados 55
6.5. Fase 3: Introducción de información ontológica en el modelo predictivo 56
6.5.1. Experimento 1: Predicción de relaciones en Freebase empleando WordNet como ontología 56
6.5.1.1. Metodología 57
6.5.1.2. Generación de la información ontológica 58
6.5.1.3. Resultados 60
6.5.2. Experimento 2: Empleo de las relaciones pseudo-ontológicas de WordNet como información ontológica y predicción sobre el resto de relaciones 60
6.5.2.1. Metodología 61
6.5.2.2. Resultados 62
6.5.3. Experimento 3: Predicción sobre relaciones en Freebase empleando la ontología de DBpedia 63
6.5.3.1. Preprocesado de los datos 64
6.5.3.2. Metodología 68
6.5.3.3. Resultados 69
6.6. Fase 4: Predicción sobre hechos que incluyen entidades nuevas 70
6.6.1. Metodología 71
6.6.2. Resultados 72
7. CONCLUSIONES Y TRABAJO FUTURO 75
7.1. Conclusiones 75
7.2. Trabajo Futuro 76
A. ANEXO A. Implementación del modelo *Neural Tensor Network* en Pytorch 87
B. ANEXO B. Implementacion de la función de pérdida del modelo *Neural Tensor Network* en Pytorch 91

Índice de figuras

1. Organización en fases del trabajo 4
2. Ejemplo de red de neuronas autocodificable o *autoencoder* 11
3. Diferencia entre las redes de neuronas *feed-forward* y las redes de neuronas recurrentes. Fuente: [2] 12
4. Matriz de co-ocurrencias obtenida con un tamaño de ventana w=2 . . 21
5. Representación de la obtención de *word embeddings* por el modelo de Word2Vec. 23
6. Ejemplo de grafo de conocimiento y conversión del mismo en tripletas o hechos 25
7. Resultado de búsqueda de un elemento en Google con la información extraída del Google Knowledge Graph. 27
8. Ejemplo de tensor $\mathbf{Y}$. Fuente: [71]. 31
9. Modelo de tensor empleado en RESCAL. Fuente: [72]. 31
10. Ejemplo de predicción de nuevas relaciones entre entidades existentes empleando RESCAL. Fuente: [72]. 32
11. Visualización de la red de neuronas tensorial. Cada caja representada con una línea discontinua representa un nivel de profundidad (número de rodajas) del tensor. En este caso, el tensor tiene profundidad 2. Fuente: [86]. 33
12. Visión general del funcionamiento del modelo, así como de la generación de los vectores de entidades. Fuente: [86]. 35
13. Diagrama explicativo del modelo TransR. Fuente: [57]. 36
14. Comparación entre las relaciones en una ontología *vs* en una base de conocimiento. 39
15. Extracción de patrones de una relación combinando información ontológica con la información extraída de la base de hechos. 41
16. Proceso de obtención de los vectores de entrada del modelo incorporando información ontológica. 44
17. Representación de la iteración de entrenamiento realizada por la red de neuronas tensorial. 48
18. Proceso de obtención de la información ontológica de las entidades del subconjunto empleado. 58
19. Diagrama explicativo del experimento realizado sobre *WordNet*. En rojo, se representan las relaciones empleadas para obtener la información ontológica, y en azul, las relaciones a predecir. 61
20. Esquema general del proceso seguido para generar los vectores de las entidades de *Freebase* con información ontológica extraída de DBpedia. 65
21. Diagrama explicativo del proceso de razonamiento del modelo dado un hecho compuesto por una entidad conocida y una entidad nueva. . 70

Índice de cuadros

1. Porcentaje de incompletitud para algunas relaciones asociadas a entidades de tipo *PERSONA* en la base de hechos Freebase. *A la izquierda:* Porcentaje de incompletitud para todas las entidades *PERSONA* existentes en la base de conocimiento. *A la derecha:* Porcentaje de incompletitud para las 100K entidades más frecuentes del tipo *PERSONA* existentes en la base de conocimiento. Fuente: [97]. 37
2. Relación del número de relaciones y de entidades representadas para *Wordnet* y *Freebase*. También se indica el número de tripletas de entrenamiento, validación y test por cada conjunto. Fuente: [86]. . . . 45
3. Especificación de los parámetros de la red de neuronas tensorial. . . . 47
4. Parámetros empleados por Socher et al. [86] para el entrenamiento del modelo. 51
5. Resultados obtenidos por el modelo para el conjunto completo de relaciones de *WordNet* 53
6. Métricas obtenidas por el modelo para el conjunto de datos de WordNet 53
7. Resultados obtenidos por el modelo para el conjunto de relaciones de *Freebase*. Las relaciones marcadas como N/A son aquellas para las que los autores no han incluido ejemplos de validación ni de evaluación. 54
8. Métricas obtenidas por el modelo para el conjunto de datos de Freebase 54
9. Número de tripletas resultantes por relación tras la división del conjunto total de hechos en los tres conjuntos de entrenamiento, validación y test. En los ejemplos de validación y evaluación se incluyen tanto hechos correctos como incorrectos. 57
10. Resultados obtenidos en las cinco pruebas realizadas. 60
11. Especificación del número de tripletas por relación empleadas en el experimento 6.5.2. 62
12. Resultados obtenidos por ambos modelos (con y sin información ontológica) para los conjuntos de datos de test de *Wordnet* para cada una de las siete relaciones a predecir. 62
13. Resultados obtenidos por el modelo empleando y sin emplear información ontológica para el conjunto de test de *Freebase* por cada una de las seis relaciones a predecir 69
14. Resultados obtenidos por el modelo para el conjunto compuesto únicamente por entidades conocidas y para el conjunto compuesto tanto por entidades conocidas como por entidades nuevas. 72

1. INTRODUCCIÓN

1.1. Introducción

La representación del conocimiento ha sido, históricamente, uno de los retos principales de la Inteligencia Artificial. Una de las formas más utilizadas para modelizar este conocimiento de forma coherente y estructurada son las ontologías. Una ontología [73] representa una definición formal de tipos, propiedades y relaciones entre entidades aplicadas a un dominio concreto. Este tipo de representaciones resultan muy intuitivas para el ser humano, además de ser también fácilmente trasladables al lenguaje informático. Una de las características más interesantes de esta forma de modelización es que está jerarquizada. En una ontología, se realiza una distinción entre los elementos volátiles, que no se mantienen en el tiempo: las instancias o recursos; y los elementos que se mantienen invariantes: los conceptos o clases. Por ejemplo, el vino *Chateau-Morgon-Beaujolais* es una subclase de la clase *Beaujolais*, que a su vez es una subclase de la clase *Red Wine* [73].

De esta forma de representar el conocimiento aparecen en la década de 1970 las bases de conocimiento o *Knowledge Bases*, que emplean la estructura y la metainformación proporcionada por la ontología para representar el conocimiento, en este caso instancias, de una manera estructurada. A partir del conocimiento representado en estas bases, empleando los llamados motores de inferencia, es posible responder preguntas e inferir nuevo conocimiento. Estas bases de conocimiento, al estar generadas de acuerdo a una ontología, recogen conocimiento estructurado de un dominio concreto. Así, existen bases de conocimiento como *Wordnet*[23], que representa qué entidades son similares a otras a nivel semántico (sinonimias), o qué conceptos representan una parte de otro (por ejemplo, el concepto ojos sería una parte del concepto cara).

La combinación de las bases de conocimiento con los motores de inferencia es referida también como sistemas basados en el conocimiento o *Knowledge-Based Systems*. Uno de los primeros sistemas de este tipo es *Mycin* [84], un sistema que representa el conocimiento con incertidumbre mediante un conjunto de reglas a las que se les otorga un factor de certeza. Sin embargo, este tipo de sistemas basados en reglas son poco robustos, por lo que han ido siendo sustituidos progresivamente por otro tipo de sistemas más eficientes, siendo en la actualidad las redes Bayesianas [25] la forma de representar e inferir conocimiento con incertidumbre más utilizado actualmente.

Empleando esta idea de los sistemas basados en el conocimiento, Google lanza en el año 2012 el llamado *Google Knowledge Graph* [31]. Este grafo busca tener un dominio universal, representando así todas las entidades existentes y las relaciones entre las mismas, sin estar sujetas a un único contexto. Al tener un dominio tan amplio, tiene una gran complejidad, y está incompleto. Resulta complejo introducir una nueva entidad y determinar con qué otras entidades se relaciona y qué tipo de relación las une. Esta tarea no está automatizada, sino que son los propios usuarios quienes introducen nuevas entidades al grafo y determinan con qué otras entidades se relacionan y cómo.

En este trabajo, se trata de dar solución a este problema empleando técnicas de aprendizaje automático. Se pretende construir un modelo escalable que sea capaz de determinar, dadas dos entidades existentes, el tipo de relación existente entre ellas, además de, explotando la información jerárquica proporcionada por las ontologías, ofrecer la posibilidad de, dada una entidad conocida y una entidad nueva, determinar la forma en que éstas se relacionan.

1.2. Motivación

La principal motivación de este trabajo es buscar una aproximación que reduzca el problema de incompletitud existente en los grafos de conocimiento, al ser este uno de los problemas que más limitan la explotación de este tipo de modelos. En la actualidad, no existe una forma automática de introducir una nueva entidad en el grafo y que éste de manera automática determine con qué entidades existentes se relaciona y cómo. Por tanto, tratar de dar solución a este problema supone un importante avance. Para llevar a cabo esta tarea, se parte de la hipótesis de que la introducción de información ontológica sobre cada una de las entidades, facilita la introducción de entidades nuevas en el grafo.

Esto supone conseguir trasladar las restricciones existentes de manera explícita en las relaciones entre tipos de una ontología a un modelo de aprendizaje automático. Por ejemplo, en una ontología la relación *ganar un premio* se da explícitamente entre una persona y un premio. Sin embargo, trasladar esta restricción al área del aprendizaje automático no resulta tan trivial, ya que las entidades no se representan de manera explícita. Introducir esta información explícitamente al modelo supone que, de alguna forma, él mismo infiriera esa restricción.

1.3. Objetivos

El primer de este trabajo es mostrar que se mejora la capacidad de predicción de las relaciones entre dos entidades mediante la introducción de información ontológica explícita. Adicionalmente, se pretende predecir relaciones en las que una de las entidades es desconocida por el modelo.

A fin de lograr dichos objetivos, se propone una división del trabajo en fases, tal y como se refleja en el organigrama 1:

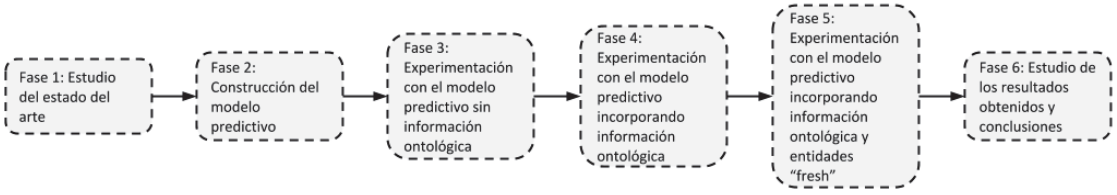


Fig. 1: Organización en fases del trabajo

A continuación, se detalla el trabajo realizado en cada una de las fases:

- **Fase 1:** Estudio de los trabajos relacionados.
  - Estudio de los trabajos relacionados con el aprendizaje automático, concretamente con las redes de neuronas artificiales
  - Estudio de los trabajos relacionados con representación distribuida de entidades o *word embedding*.
  - Estudio de los trabajos relacionados con la compleción de bases de conocimiento o *Knowledge Base Completion*.
  - Discusión de las conclusiones extraídas de los mismos y las limitaciones encontradas.
- **Fase 2:** Construcción del modelo predictivo a emplear.
- **Fase 3:** Experimentación en predicción de relaciones sin incorporar información ontológica en las entidades.
- **Fase 4:** Experimentación en predicción de relaciones introduciendo información ontológica en las entidades:
  - Empleando una ontología creada de manera manual sin introducción de entidades nuevas.
  - Empleando una ontología existente sin introducción de entidades nuevas.
- **Fase 5:** Experimentación con el modelo predictivo incorporando introducción ontológica y entidades nuevas ("*fresh*")
- **Fase 6:** Análisis de los resultados obtenidos y extracción de conclusiones. Planteamiento de trabajos futuros.

1.4. Estructura del documento

El presente documento se estructura de la siguiente manera. En primer lugar, se realiza una exposición de las tecnologías empleadas, así como de los trabajos relacionados. A continuación, se plantea de forma detallada cuál es el problema a resolver, puntualizando las dificultades encontradas de cara a la resolución del mismo para, posteriormente, exponer detalladamente el proceso de resolución junto con sus correspondientes resultados.

El documento finaliza con una descripción de las conclusiones generales del trabajo, así como un planteamiento de las posibles líneas futuras que se podrían generar a partir de él.

2. REDES DE NEURONAS ARTIFICIALES

Las redes de neuronas artificiales son uno de los métodos de aprendizaje automático que gozan de mayor popularidad en la actualidad. Estos modelos matemáticos tienen su inspiración en el comportamiento observado en las neuronas del cerebro humano, y reciben también el nombre de modelos conexionistas. Son capaces de "aprender" a realizar tareas directamente a partir de conjuntos de datos, sin necesidad de intervención previa; es decir, son capaces de llevar a cabo abstracciones, y de extraer patrones a partir de conjuntos de datos, lo que les permite resolver problemas particularmente complejos.

Los componentes principales de las redes de neuronas son: neuronas, estado de activación, función de salida, función de activación, patrón de conexiones y regla de propagación

Neuronas

Las neuronas son las unidades de procesamiento de los datos de la red. Estos elementos, capaces de realizar operaciones sencillas, poseen un estado interno, que es capaz de cambiar en función de la señal que reciben.

Su funcionamiento es simple: cada neurona recibe las entradas de las neuronas vecinas o del exterior y, tras realizar una operación matemática, obtiene un valor de salida que envía al exterior o a las neuronas con las que se encuentra conectada. Este proceso se realiza de manera paralela en todas las neuronas. Dentro de las neuronas, se distinguen tres tipos distintos: neuronas de entrada, ocultas y de salida. Las neuronas de entrada son aquellas que reciben la información del exterior del sistema, las neuronas de salida envían la información al exterior y las neuronas ocultas son aquellas cuyas entradas y salidas de información se encuentran dentro del sistema, no teniendo así contacto con el exterior.

Estado de activación

Las neuronas tienen un estado de activación que cambia a lo largo del tiempo. Durante cada instante de tiempo t, la neurona se encuentra en un estado del conjunto ε. Dicho estado puede pertenecer a un conjunto binario $\varepsilon = \{0,1\}$, donde 0 es el estado de inhibición y 1 el de activación; o un intervalo continuo de valores, por ejemplo $\varepsilon = [0,1]$.

Función de salida

Las neuronas, entre ellas, se transmiten la información en forma de señales. La potencia de una señal depende del grado de activación de la neurona. Para obtener dicha señal de salida, la función de salida obtiene este valor a partir del estado de activación de la neurona. Generalmente, se suele emplear la función identidad, por lo que la salida es el estado de activación de la neurona.

Patrón de conexiones

Además de las neuronas, la otra parte fundamental en la arquitectura de la red son las conexiones que se establecen entre ellas. A nivel conceptual, una red de neuronas puede ser vista como un grafo dirigido, en el cual los nodos son las neuronas y los arcos, las conexiones entre ellas. Cada arco tiene asociado un valor que indica el peso de dicha conexión.

Al conjunto de pesos de las conexiones de una red de neuronas se les denomina parámetros de la red.

Regla de propagación

Dadas las salidas de las neuronas y la matriz de conexiones, se necesita una regla que, a partir de estos valores, determine cuáles serán los valores de entrada de las neuronas siguientes. A esta regla se la denomina regla de propagación, y la usada típicamente realiza, por cada neurona, un sumatorio de todas sus entradas multiplicadas por los pesos de las conexiones de cada una. Al resultado de esta operación se la denomina *net* o entrada neta o total.

Función de activación

Para cada neurona, su estado de activación se calcula aplicando la función de activación a la entrada neta que recibe la neurona. El valor devuelto por esta función será el nuevo estado que tome la neurona. Pueden emplearse varios tipos de funciones, dependiendo de si el espacio de estados de activación es binario o no. En el caso de los estados de activación no binarios, las funciones más usadas tradicionalmente son la sigmoide y la tangente hiperbólica (funciones logísticas), al ser no sólo fáciles de derivar, sino biológicamente plausibles.

2.1. Aprendizaje

Para conseguir que una red aprenda a resolver el problema deseado, es necesario entrenarla para que sea capaz de, dada una información de entrada, devolver la respuesta esperada para la misma. Conseguir que la red aprenda consiste en obtener valores óptimos en los parámetros del modelo (el conjunto de pesos de la red), de manera que al ponderar las entradas de la neurona por los pesos obtenidos tras el entrenamiento, éste sea capaz de devolver una respuesta adecuada.

En función a la naturaleza del problema a resolver, se distinguen dos tipos de aprendizaje: aprendizaje supervisado y aprendizaje no supervisado.

Aprendizaje supervisado

En este tipo de aprendizaje, se le presenta a la red un conjunto de datos de entrenamiento, compuesto por duplas formadas por un vector de estímulos y su correspondiente respuesta correcta. Este conjunto de entrenamiento debe ser suficientemente representativo como para que la red aprenda toda la información necesaria.

En este caso, durante el proceso de entrenamiento, se compara la salida obtenida por la red con la salida esperada, y se reajustan los pesos de las conexiones de manera que, ante la próxima visualización del mismo patrón, el modelo sea capaz de dar la respuesta correcta. Este proceso se realiza múltiples veces con cada uno de los vectores de entrenamiento. El entrenamiento finaliza cuando la red sea capaz de devolver la respuesta correcta para todos los datos de entrenamiento presentados. En ese caso se dice que la red ha convergido a una solución.

Ya que este tipo de aprendizajes tienen un objetivo definido, dicho objetivo debe ser definido mediante una función, que calcule, de acuerdo al problema, cómo de diferente es la salida obtenida de la deseada, es decir, cual es el error cometido. Esta función, que es minimizada por el modelo durante el proceso de entrenamiento, se denomina función de pérdida.

El algoritmo que regula cómo y cuánto deben variar los pesos es el algoritmo de retropropagación. Se basa en el uso del gradiente de los parámetros de la red respecto a la función de pérdida para obtener cuáles deben ser los nuevos pesos de la red. El gradiente de una función indica cuál es la dirección del mínimo; por tanto, dado que durante el entrenamiento se busca minimizar el valor del error, resulta muy conveniente emplear el gradiente para saber hacia qué valores deben variar los pesos a fin de alcanzar el mínimo. En el algoritmo de retropropagación clásico [80], se itera de manera ordenada sobre cada una de las tuplas de entrenamiento, calculando en cada iteración para cada pareja de atributos-etiqueta cuál es el gradiente del error, y actualizando los pesos de la red. Para determinar cuánto deben modificarse los pesos del modelo respecto al gradiente de error obtenido se emplea un hiperparámetro denominado tasa de aprendizaje o *learning rate*. El problema principal de este tipo de entrenamiento es que el correcto aprendizaje está condicionado por el orden en el que se presentan los datos al modelo. Para introducir una componente de aleatoriedad que solucione esta carencia, se propone una variante del algoritmo de retropropagación, el algoritmo de descenso del gradiente estocástico, o *SGD* [47]. En este algoritmo, los ejemplos de entrenamiento a procesar en cada iteración son escogidos de manera aleatoria. Aunque esta aproximación ofrezca un mejor funcionamiento que el algoritmo de retropropagación clásico, no siempre es capaz de converger a una solución óptima, ya que al modificar los pesos por cada elemento del conjunto de datos de entrenamiento, se producen grandes oscilaciones. Una solución a esto es emplear un entrenamiento por lotes o *batch*. En este tipo de entrenamiento se procesa cada ejemplo de entrenamiento y se obtiene el gradiente del error para cada una de ellas. Dichos gradientes se van acumulando y, tras procesar todos los datos de entrenamiento, se lleva a cabo la actualización de los pesos. Al tener que procesar todos los datos antes de actualizar los pesos, esta forma de entrenamiento tiene una convergencia mucho más estable. Sin embargo, resulta excesivamente lenta, por lo que se propone que, en vez de procesar el conjunto de datos completo antes de actualizar, se procese un número fijo de ejemplos de entrenamiento, y que tras procesar dicho número de ejemplos se lleve a cabo la actualización de pesos. El número de ejemplos a procesar se define por un hiperparámetro, el tamaño de batch, y es fijo durante todo el proceso de entrenamiento. Este tipo de entrenamien-

to, denominado entrenamiento por mini-batch, obtiene unos resultados similares al entrenamiento por lotes, pero con la ventaja de ser significativamente más rápido.

Aprendizaje no supervisado

En este caso, el conjunto de entrenamiento está únicamente compuesto por datos de entrada, pero no se especifica cuál es la respuesta correcta a dichas entradas, ya que no se busca comparar si la respuesta por la red es correcta o no. Lo que se busca en este modelo de aprendizaje es que, a partir de un gran conjunto de datos, la red sea capaz de construir sus propias asociaciones, infiriendo patrones de ellos.

Para ello, la cantidad de datos que se necesita es mucho mayor que en el aprendizaje supervisado y, dado que ambos tipos persiguen distintos objetivos, los procedimientos de aprendizaje que se emplean son distintos.

2.2. Redes de neuronas alimentadas hacia delante, o feed-forward

En este tipo de modelos, las neuronas se organizan en capas, de manera que las neuronas de una misma capa no se conectan entre sí, sino que se conectan con las neuronas de otras capas. Además, las conexiones son siempre dirigidas hacia delante, de manera que cada neurona de una capa sólo envía información a las neuronas de la capa siguiente y, de igual manera, sólo recibe información de las neuronas de la capa anterior.

Dentro de este tipo de modelos, cabe destacar las redes de neuronas autocodificables o *autoencoders* [39] [7] [16]. En este tipo de modelos, el objetivo es obtener una codificación de los valores de entrada que sea altamente representativa y que tenga una dimensión determinada, para luego ser empleada en otro tipo de tareas, tales como clasificación. En la figura 2 se muestra un ejemplo de autoencoder. Como se observa, se trata de un modelo con arquitectura simétrica, con dos fases muy diferenciadas. Ya que el objetivo de los *autoencoders* es obtener la misma representación de los datos de entrada en la salida, se produce una primera fase de codificación de los datos, que abarca desde la capa de entrada hasta la capa central oculta. Durante este proceso, se va reduciendo paulatinamente la dimensión de los datos. En la segunda fase, se lleva a cabo la decodificación, en la cual, partiendo de las representaciones de dimensionalidad reducida obtenidas tras la primera fase, se trata de reconstruir los datos originales de entrada.

La particularidad de estos modelos es que la información relevante no se obtiene a la salida, sino que se extrae de la capa oculta central, ya que es allí donde se encuentran las representaciones deseadas de los datos.

2.3. Redes de neuronas recurrentes

Las redes de neuronas recurrentes (RNN) son un tipo especial de redes de neuronas las que las neuronas no se organizan en capas [9]. De esta manera, una neurona

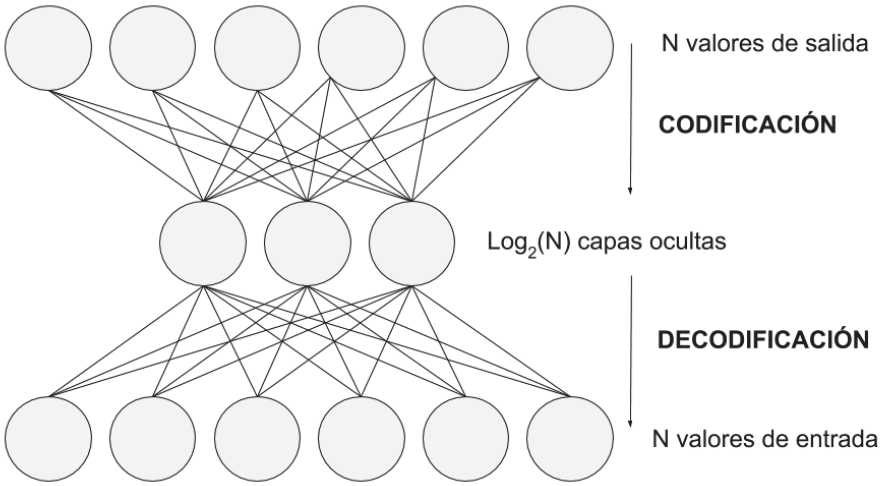


Fig. 2: Ejemplo de red de neuronas autocodificable o *autoencoder*

puede conectarse con cualquier otra e, incluso, consigo misma. En la figura 3 se muestra la comparativa entre las arquitecturas alimentadas hacia delante y las redes de neuronas recurrentes. Como se observa en la figura, en las RNN aparecen ciclos en las conexiones, lo que significa que para calcular el valor de salida hay que tener en cuenta, además del estado actual, el valor de salida obtenido anteriormente. Esto permite que la red tenga 'memoria'. Por esta razón, este tipo de redes se utilizan en tareas en las cuales los datos de entrada presentan dependencias entre sí, como el reconocimiento del habla. Este tipo de modelos, debido a sus particularidades, resultan más complejos de entrenar que los modelos alimentados hacia delante.

Dentro de las redes de neuronas recurrentes, existen múltiples variantes, tales como las redes de neuronas recursivas, las cuales permiten trabajar con entradas estructuradas y devolver salidas con la misma estructura. Un ejemplo clásico es representar a la entrada una información estructurada en formar de árbol, de manera que la salida sea también una estructura de árbol.

Otro tipo son las redes de neuronas recurrentes bidireccionales, en las cuales se conectan dos capas ocultas con direcciones opuestas a la misma salida, permitiendo que para calcular el valor de salida se tenga en cuenta la información no sólo acerca de los estados pasados, sino de los estados futuros. Este tipo de modelos son particularmente útiles para tareas en las que es relevante el contexto de la entrada, tales como reconocimiento de lenguaje escrito.

2.4. Evolución de las redes de neuronas

La primera aproximación a las redes de neuronas se produce en 1943, cuando McCulloch y Pitts [59] proponen un primer modelo de neurona artificial que busca

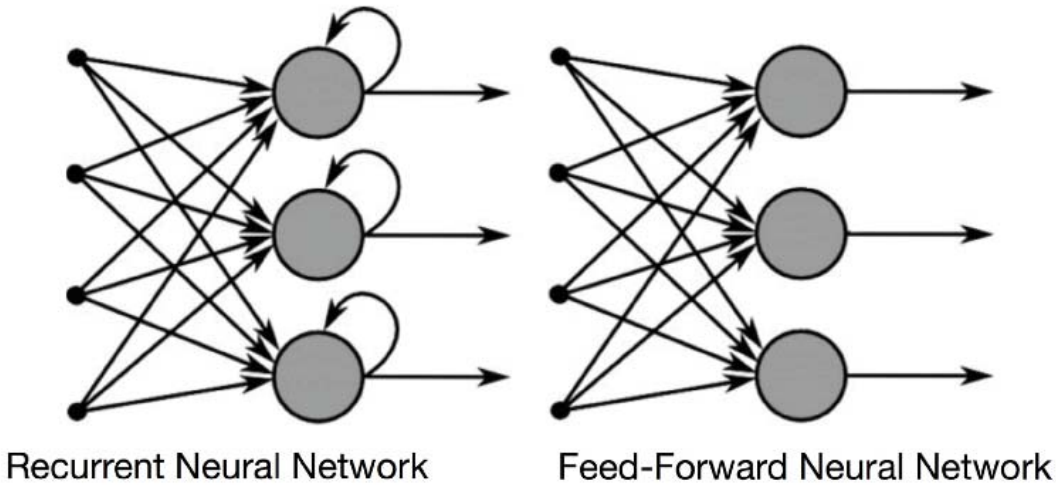


Fig. 3: Diferencia entre las redes de neuronas *feed-forward* y las redes de neuronas recurrentes. Fuente: [2]

simular el comportamiento de las neuronas del cerebro. En estas primeras neuronas, los pesos se inicializan manualmente, y se emplea una función de activación umbral. A partir de este momento, se plantean dos enfoques distintos en el uso de estos algoritmos. Por un lado, una corriente que se centra en el estudio de los procesos biológicos del cerebro (*biological neural networks* [56]), y otro centrado en las aplicaciones de dichos algoritmos (*artificial neural networks*).

Dentro de la corriente de las redes de neuronas artificiales, el siguiente hito tiene lugar en 1958, cuando Frank Rosenblat [79] introduce el perceptrón, un primer acercamiento a lo que serían las redes de neuronas, compuesto únicamente de una capa de entrada y una capa de salida, capaz de resolver problemas de clasificación linealmente separables. Sin embargo, este modelo no es capaz de realizar operaciones tan sencillas como el or-exclusivo, por lo que en 1969, Minsky y Papert publican el libro *Perceptrons* [64], en el cual ponen de manifiesto estas carencias del perceptrón. Además de esta crítica, proponen la hipótesis de que aumentar el tamaño del perceptrón permitiría resolver problemas más complejos, pero que el método de aprendizaje propuesto por Rosenblatt no es capaz de funcionar correctamente en ese caso. Esto hace que las redes de neuronas, durante un gran periodo de tiempo, caigan en desuso.

En el año 1986, Rumelhart publica el algoritmo de retropropagación del gradiente del error [81]. Este algoritmo permite entrenar redes de neuronas artificiales mediante la propagación del error existente en la salida respecto al valor esperado, trasladando ese error hacia la entrada y permitiendo que el modelo cambie para ajustar la salida obtenida a la esperada. Esto supone un importante avance, ya que este método permite entrenar redes de neuronas con capas ocultas y funciones de activación no

lineal.

Solucionado el problema del entrenamiento, el siguiente paso es estudiar hasta qué punto es cierto que un modelo más complejo que el perceptrón permite resolver problemas de carácter no lineal. Poco después, se demuestra que el perceptrón multicapa, es decir, una red de neuronas, con funciones de activación no lineales, compuesta simplemente por una entrada, una o más capas ocultas y una salida es un aproximador universal [42]. Esto implica que, a nivel teórico, estos modelos pueden aproximar, con cierto nivel de precisión, cualquier tipo de función, independientemente de si el carácter de la misma es lineal o no.

Esto provoca un resurgimiento de las redes de neuronas, y se empiezan a utilizar también para llevar a cabo tareas de aprendizaje no supervisado [39] [7] [16], además de expandir su uso a áreas como la visión por computador [55], o los sistemas de ayuda a la decisión [20][4].

Otra de las áreas donde cobran también gran importancia las redes de neuronas es en el reconocimiento automático del lenguaje, donde inicialmente se propusieron las llamadas *Time-Delay Neural Networks*, o TDNN; un tipo de modelos *feed-forward* en los cuales se aplica un cierto tiempo de retardo en las conexiones de la red para mantener ciertos datos en ella durante mayor tiempo. Fueron sustituidas posteriormente por las redes de neuronas recurrentes actuales [9], debido a que, mientras que en las TDNN es el diseñador del modelo el que debe determinar de manera manual los tiempos de retardo de cada una de ellas, en las redes de neuronas recurrentes gracias a los ciclos existentes en las conexiones, todo el proceso de mantenimiento del estado de memoria dentro del modelo se realiza de manera automática.

2.5. Deep Learning

Existiendo ya sistemas de cómputo suficientemente robustos, y teniendo ya un algoritmo de entrenamiento, empiezan a aparecer modelos con un número elevado de capas, los cuales, tal y como teorizaron Minsky y Papert [64], son capaces de resolver problemas más complejos. Sin embargo, este incremento del número de capas provoca la aparición de múltiples problemas que hasta el momento no se habían contemplado. Esta nueva corriente de emplear modelos de una gran dimensión y, especialmente, de proponer soluciones a los problemas que dichos modelos plantean, es conocida como *Deep Learning* o aprendizaje profundo [37]. El aprendizaje profundo es el término que se refiere al conjunto de algoritmos de aprendizaje automático que permiten obtener soluciones a problemas particularmente complejos, para lo que se necesita un gran número de datos. En el caso concreto de las redes de neuronas, el término *deep* se refiere a aquellas arquitecturas que tienen un tamaño elevado.

Lo primero que se observa es que, si bien es cierto que incrementar el tamaño de la red permite resolver problemas más complejos, se observa que, superado cierto umbral, una red entrenada por el algoritmo de retropropagación no es capaz de aprender correctamente. Este problema es conocido como el problema de desvanecimiento o explosión del gradiente, o problema fundamental del aprendizaje profundo. El problema de desvanecimiento se produce en las redes alimentadas hacia delan-

te, mientras que el problema de explosión se produce en las redes recurrentes. El problema del desvanecimiento del gradiente sucede debido a que, en el algoritmo de retropropagación [81], el gradiente del error se propaga desde las capas finales a las capas superiores, reduciéndose de manera progresiva tras pasar por cada una de las capas. Si el gradiente de dicho error en las capas finales es muy cercano a cero, al propagarse a las capas superiores será cada vez más cercano a cero, provocando que la red deje de modificar sus pesos y que, consecuentemente, deje de aprender. En las redes recurrentes sucede el problema inverso y es que, ya que las neuronas pueden tener ciclos, el gradiente se acumula, por lo que si este gradiente es demasiado alto, provoca que los valores de los pesos aumenten de manera descontrolada, haciendo que el modelo no converja hacia una solución. El problema de la explosión del gradiente en las redes de neuronas recurrentes se resuelve con de las LSTM (*Long-Short Term Memory*) [40].

En las redes de neuronas alimentadas hacia delante, el problema del desvanecimiento del gradiente del error se puede reducir mediante el empleo de la función de activación *Rectified Linear Unit*, o ReLU [36][30], representada en la función (1).

$$f(x) = max(0, x) \tag{1}$$

Las funciones de activación empleadas clásicamente son la tangente hiperbólica y la función sigmoide, las cuales obtienen buenos resultados para redes pequeñas. Sin embargo, cuando el número de capas aumenta en exceso, estas funciones no obtienen buenos resultados, ya que convergen de manera prematura, cayendo en óptimos locales. Esto es provocado porque las derivadas de ambas funciones en los extremos es prácticamente cero, lo que provoca que el gradiente sea cercano a cero y, consecuentemente, de lugar al problema de desvanecimiento del gradiente. Sin embargo, la función ReLU, además de ser significativamente más sencilla que las anteriores, no se satura. Existen diferentes variantes de ReLU, como la *Leaky ReLU* [58], la *Parametric ReLU* [34] o la *ELU* [19].

Otro de los problemas principales que aparecen al aumentar el tamaño de los modelos es el problema del *overfitting*, o sobreajuste. Las redes de neuronas necesitan un gran número de datos para aprender y lograr un nivel de abstracción lo suficientemente alto como para que el modelo sea capaz de generalizar y funcionar correctamente con datos similares a los que se ha entrenado. Si el número de datos de entrenamiento de la red es bajo o el modelo es excesivamente grande, entonces memoriza los datos de entrada y no es capaz de extraer patrones de ellos, por lo que no responde correctamente con nuevos datos. Sin embargo, conseguir una gran cantidad de datos o encontrar un modelo de tamaño adecuado no siempre es fácil. En general, la forma de proceder actualmente es construir una red excesivamente grande y aplicar técnicas que eviten el sobreentrenamiento. Existen varias técnicas de este tipo, desde las más simples como interrumpir prematuramente el entrenamiento o aumentar el número de datos artificialmente, a otras más sofisticadas, como la regularización L1 y L2 [69]. En la regularización L1 y L2, se añade un término a la función de error, el cual impone una penalización al modelo que restringe los grados de libertad de los parámetros de la red. Otra de las técnicas existentes para evitar

el sobreajuste es la técnica de *dropout* [38]. Esta técnica consiste en eliminar, de manera aleatoria, ciertas neuronas durante el proceso de entrenamiento, eliminando así parte de su capacidad de cómputo.

La inicialización adecuada de los pesos de la red favorece significativamente el aprendizaje. En este área, destaca el trabajo de Xavier Glort, que propone el algoritmo de inicialización Xavier [29], el cual tiene en consideración la posición de la capa a inicializar, dando magnitudes distintas a los pesos de las capas más cercanas a la entrada respecto a los de las capas más cercanas a la salida. Esto reduce significativamente la posibilidad de aparición del problema de desvanecimiento o explosión del gradiente respecto a la inicialización aleatoria. La otra forma más habitual de inicialización es emplear una función Gaussiana, aunque se ha comprobado que, por lo general, el algoritmo Xavier ofrece un mejor funcionamiento, ya que ayuda a evitar la caída en óptimos locales.

Otra de las dificultades que plantea el aprendizaje profundo es que, además de necesitar grandes conjuntos de datos, se necesitan también sistemas de cómputo muy potentes. Una red profunda necesita operar con matrices de millones de dimensiones, por lo que se necesitan sistemas con una alta capacidad computacional. Además, debido a la profundidad de las mismas, el tiempo necesario para entrenar una red puede llegar a ser muy elevado. Tanto el tiempo de entrenamiento requerido como la capacidad de cómputo necesaria crecen de manera lineal con el tamaño de la red. Por ello, se necesitan procesadores con una mayor capacidad de procesamiento de matrices que una CPU normal.

En el año 2005, se propone el uso de las unidades de procesamiento gráfico, o GPUs, para ser utilizadas para entrenar y manejar algoritmos de aprendizaje automático [88]. La particularidad de estos sistemas es que llevan a cabo computación en paralelo y, al estar pensados para procesamiento de imágenes, que son matrices de gran dimensión, resultan muy eficientes para procesar bloques de datos de gran tamaño a una alta velocidad. Inicialmente, estos sistemas se utilizaban únicamente para procesamiento gráfico, hasta que surgieron las GPGPU (*General-Purpose Graphical Processing Units*). Estas unidades permiten llevar a cabo tareas de cómputo que típicamente lleva a cabo el procesador pero de manera paralelizada, provocando así no sólo un reparto de carga de trabajo entre GPU y CPU, sino consiguiendo que la tarea se lleve a cabo mucho más rápido. En una primera prueba de uso de computación GPGPU para entrenar modelos de *deep learning*, se estimó que esta forma de procesamiento era capaz de reducir el tiempo de cómputo en cerca de 70 veces [76] el tiempo que se tardaba habitualmente con una CPU.

En la línea de optimizar el proceso de entrenamiento, se plantean también nuevas variaciones del algoritmo de retropropagación. Estos algoritmos permiten acelerar el proceso de aprendizaje del modelo, ayudándole a converger más rápido hacia la solución. Una posible solución es la introducción de la inercia, o *momentum* [89]. Este concepto, basado en su homónimo en el campo de la física, busca reducir la influencia de las oscilaciones en el proceso de entrenamiento. Para ello, una vez obtenidas las actualizaciones de los pesos de la red, se añade una fracción de los valores de variación obtenidos en la iteración anterior.

A pesar de que el *momentum* permite reducir las oscilaciones y, consecuentemente, conseguir que la convergencia sea más rápida, tiene una dificultad, y es que no detecta cuándo se ha alcanzado un mínimo. Ya que a la inercia se le da un valor elevado, el cual es constante, puede suceder que cuando se alcance el mínimo de la función, no se detecte como tal, y el algoritmo se dirija a un óptimo local. Este problema fue estudiado por Nesterov [68], el cual propuso un algoritmo que lleva su nombre. Una vez computado el gradiente en la manera habitual, se calculan cuáles son los nuevos parámetros tras ser actualizados con el factor momento seleccionado. A continuación, se calcula el gradiente del error respecto de dichos parámetros futuros. Este gradiente futuro permite comprobar si los nuevos parámetros permiten acercarse al mínimo o, por el contrario lo alejan, lo que permite corregirlos si se da el segundo caso mediante la modificación del *momentum* en la siguiente iteración.

Otras aproximaciones no consideran el uso del factor momento, y se centran en adaptar la tasa de aprendizaje para asegurar la convergencia del modelo. Un ejemplo es el algoritmo AdaGrad [22], el cual utiliza un valor distinto de tasa de aprendizaje para cada parámetro en cada iteración, basándose en los gradientes previos obtenidos para dichos parámetros. Esto hace que las actualizaciones de los parámetros ante los patrones de entrada infrecuentes sean más bruscas que para los frecuentes. Este algoritmo funciona particularmente bien para problemas en los que los datos son dispersos, aunque presenta una gran desventaja, y es que la tasa de aprendizaje solamente decrece. Ya que este parámetro regula cuánto deben actualizarse los parámetros en función del gradiente, cuando sea cercano a cero, los pesos de las conexiones dejarán de actualizarse, deteniéndose así el entrenamiento. Para solucionar este problema se propone el algoritmo AdaDelta [101], una adaptación de AdaGrad que elimina el problema de decrecimiento constante de la tasa de aprendizaje. Sin embargo, al no considerar el factor momento, este algoritmo tiene una convergencia muy lenta.

Finalmente, una de las aproximaciones más recientes y más utilizadas es el algoritmo Adam [48]. Adam es el acrónimo de *Adaptative Moment Estimation*, y se trata de un algoritmo que, de igual manera que AdaGrad, emplea una tasa de aprendizaje adaptativa, en este caso sí se considera la inercia, la cual también es adaptativa, al igual que en el algoritmo Nesterov. Tener un algoritmo que emplea tanto un factor inercia como una tasa de aprendizaje adaptativos se traduce en obtener las ventajas derivadas de ambos. De esta manera, dicho algoritmo elimina el problema del decrecimiento constante de la tasa de aprendizaje, converge relativamente rápido, no presenta oscilaciones y se mantiene estable.

Todos estos avances tanto a nivel software como a nivel hardware propiciaron la consolidación de las redes de neuronas en la comunidad científica como uno de los paradigmas más importantes dentro del área del aprendizaje automático. Este interés se extiende a diversas aplicaciones. Así, en 2012 una red de neuronas convolucional, AlexNet [51], resulta vencedora en el concurso ILSVRC. Este concurso (*ImageNet Large Scale Visual Recognition Contest*), consiste en entrenar un modelo capaz de clasificar imágenes en 1000 categorías distintas. AlexNet supera de manera notable al resto de competidores, además de aumentar en un 12 % el porcentaje máximo

de acierto conseguido hasta el momento en dicho concurso. Esto atrajo el interés de grandes empresas, tales como Google, Microsoft o Facebook, que rápidamente se interesaron por el *Deep Learning* e iniciaron sus propios proyectos dentro de este área. Por ejemplo, en el concurso ILSVRC del año 2014, Google presentó su propia red de neuronas convolucional, GoogLeNet [92], que resultó vencedora ese mismo año; o Facebook, que en 2016 lanzó *DeepText* [103], un modelo profundo compuesto por redes de neuronas convolucionales y recursivas, capaz de intepretar texto incluso en imágenes. En la actualidad, son estas empresas las que continúan principalmente la investigación dentro del área del *Deep Learning*, al tener acceso a las cantidades de datos necesarias para este tipo de modelos y la capacidad de cómputo necesaria para trabajar con ellas.

Finalmente, en cuanto a los proyectos más destacados actualmente dentro del *Deep Learning*, se desarrollan principalmente en dos áreas. Por un lado, gran parte de los trabajos se dedican al área de visión por computador, donde hay que destacar el modelo *ResNet* , una red de neuronas convolucional profunda que en la actualidad representa el estado del arte en el área de reconocimiento de objetos en imágenes [33]. Por otra parte, existe una gran corriente de investigación centrada en la comprensión y tratamiento del lenguaje, tanto de forma hablada como escrita, con diferentes trabajos que buscan modelos capaces de llevar a cabo traducciones entre idiomas de manera automática [6] [83]; así como trabajos que se basan en el análisis del discurso para realizar un autocompletado del lenguaje a partir de la introducción de texto [35].

Otros campos de investigación destacados dentro del área del procesamiento del lenguaje natural en la actualidad son: la identificación de idiomas ([65] [8]), el etiquetado gramatical o *POS-tagging* [28], segmentación del habla [17], [102], [100] y pre-ordenamiento de las palabras [67], [21].

3. REPRESENTACIÓN DISTRIBUIDA DEL LENGUAJE: WORD EMBEDDING

La representación distribuida del lenguaje, más conocida por el término *word embedding* es, desde hace años, una de las principales áreas de estudio dentro del campo del procesamiento del lenguaje natural. A pesar de su potencial, la mayoría de los algoritmos de aprendizaje automático, incluyendo los algoritmos de *deep learning*, son incapaces de procesar y entender texto o palabras de manera directa. Por ello, el objetivo de este tipo de trabajos es conseguir que estos algoritmos sean capaces de procesar o "entender" el lenguaje natural para llevar a cabo tareas concretas.

Para conseguir este objetivo, lo primero es obtener representaciones vectoriales del lenguaje que sean ricas a nivel semántico, de manera que palabras cuyos significados sean similares tengan representaciones que sean similares también.

3.1. Origen

Entender y comprender el significado del lenguaje es una tarea compleja, ya que es difícil determinar qué hace representativa a una palabra. Tratando de dar solución a esta ambigüedad, en el año 1957, Firth formula la siguiente idea: una palabra está caracterizada por las palabras que la rodean [24]. Esto implica que lo más representativo de una palabra no es el significado de la misma, sino que la información más importante acerca de ella se infiere de las palabras junto a las que aparece, o lo que es lo mismo, la información contextual de la misma. La información que hace representativa a una palabra recibe el nombre de características o *features*.

Los primeros intentos de representar las características datan de la década de los 60, y uno de los más destacados es el propuesto por Osgood [66]. En este trabajo, propone una escala semántica diferencial, en la que trata de cuantificar el significado de una palabra, en concreto adjetivos y los conceptos a los que dichos adjetivos se refieren. Para ello, se le propone a un individuo dos adjetivos polarmente opuestos y se le pregunta cuál es su posición entre ellos (por ejemplo, entre "bueno" y "malo"). Con los valores dados por todos los encuestados a cada pareja de adjetivos se obtiene un valor numérico, y a partir de esos valores numéricos se obtiene el valor del concepto al que dichos adjetivos se refieren. Esa forma de representar una entidad es muy subjetiva y poco robusta, por lo que se precisan nuevas formas de representación. También en esta época, se propone el modelo de espacio vectorial [82], una forma algebraica de representar información mediante un espacio vectorial, donde los vectores están compuestos por un número fijo de identificadores. En este ámbito de representación del lenguaje, los identificadores del vector son las características de la palabra a representar.

En la década de los 90 aparecen métodos que automatizan la generación de vectores de características. Algunos de los más populares de la época son el *Latent Semantic Analysis*[53]; y otros basados en redes de neuronas artificiales, como los *Self Organizing Maps* [49][50], que mediante aprendizaje supervisado generan un espacio bidimensional de vectores de características; o las redes de neuronas recurrentes

simples (*Simple Recurrent Networks*) [41] [9].

La diferencia principal entre los modelos de *Latent Semantic Analysis* y los modelos basados en redes de neuronas recurrentes reside en el tipo de información contexual en que se basan para generar las características. Mientras que los primeros utilizan documentos completos como contexto para la palabra, los segundos utilizan otras palabras, lo cual resulta más intuitivo desde un punto de vista lingüístico. Sin embargo, las representaciones obtenidas por estos modelos tienen una dimensión muy alta, lo que hace complicado trabajar con ellas.

En los años 2000, Yoshua Bengio [10] propone una serie de trabajos que estudian cómo reducir la dimensionalidad de estas representaciones, para lo que emplea un modelo distribuido de representación de las palabras. A partir de este momento, se produce un importante impulso en el área del *word embedding*, apareciendo métodos cada vez más sofisticados y que dan lugar a representaciones cada vez mejores.

3.2. Modelos basados en conteo

Estas representaciones están basadas en la distribución semántica de las palabras, y se parte de la idea de que palabras similares tienden a aparecer juntas y, además, en contextos similares. Para delimitar el contexto de una palabra, se definen los conceptos de ventana de contexto, que es son las palabras que aparecen en una distancia w tanto a la izquierda como a la derecha de la palabra seleccionada, y de co-ocurrencia, que representa el número de veces que dos palabras han aparecido juntas en una ventana de contexto. La co-ocurrencia de cada pareja de palabras se recoge en una matriz, la matriz de co-ocurrencias, de manera que la fila i-ésima de dicha matriz será la representación de la palabra i-ésima del vocabulario. En la figura 4 se muestra el proceso de obtención de la matriz de co-ocurrencias para una ventana de tamaño 2. Habitualmente, se suelen eliminar las palabras vacías (*stopwords*), ya que no aportan información semántica.

Este tipo de modelos, a pesar de su simplicidad, cumplen con la premisa de que palabras similares tienen representaciones similares. Además, al generar representaciones vectoriales, permiten el uso de medidas de similitud, tales como la distancia coseno. La distancia coseno de las representaciones de dos palabras similares será cercana a 0, mientras que la de dos palabras opuestas será cercana a 1.

Esta forma de generar los vectores de palabras tiene un gran problema y es que no sólo la matriz de co-ocurrencias crece exponencialmente con el tamaño del vocabulario, sino que ademas dicha matriz es muy dispersa, debido a que el tamaño de ventana suele ser un valor pequeño. Para solucionar este problema se suele realizar un análisis de componentes principales (PCA) sobre la matriz de co-ocurrencias, que permite obtener representaciones de dimensionalidad reducida pero que preservan la información semántica de manera casi íntegra.

Dentro de los modelos basados en conteo, el más popular es GloVe (*Global Vectors for Word Representation*), publicado en el año 2014 por la universidad de Stanford [75]. En este modelo, se busca explicar cada palabra mediante un contexto global, de manera que para capturar el significado de una palabra, no se emplea un número

	The	Quick	Brown	Fox	Jump	Over	Lazy	Dog
The	0	1	1	0	1	1	1	1
Quick	1	0	1	0	0	0	0	0
Brown	1	1	0	1	1	0	0	0
Fox	0	1	1	0	1	1	0	0
Jump	1	0	1	1	0	1	0	0
Over	1	0	0	1	1	0	1	0
Lazy	1	0	0	0	0	1	0	1
Dog	1	0	0	0	0	0	1	0

Fig. 4: Matriz de co-ocurrencias obtenida con un tamaño de ventana w=2

limitado N de palabras, sino que se tiene en cuenta la estructura del corpus de entrada completo.

Para ello se emplea un algoritmo de clasificación no supervisada, el cual entrena empleando las estadísticas de co-ocurrencia entre las palabras del corpus, y da lugar a un espacio vectorial de representaciones con una sub-estructura altamente representativa. Esta representación preserva las similitudes entre palabras, la cual puede ser cuantificada mediante la distancia coseno.

3.3. Modelos basados en predicción

Los modelos basados en predicción emplean, generalmente, técnicas de aprendizaje supervisado. Para obtener las representaciones vectoriales del vocabulario, se plantea un problema de clasificación, en el cual existe una función objetivo a maximizar. Este objetivo que se le da es ficticio, ya que lo que se pretende es obtener las representaciones codificadas de las entradas, para luego emplear estas representaciones en una tarea distinta a la que se emplea como objetivo. El proceso general seguido por estos modelos es el siguiente:

1. Recopilar todas las instancias $t_1 \in inst(t)$ de una palabra t perteneciente a un vocabulario V

2. Para cada instancia, obtener sus palabras de contexto $c(t_1)$
3. Definir la función objetivo $score(t_i, c(t_i); \theta, E)$, cuya salida esté acotada por arriba.
4. Definir la función de pérdida $L = -\sum_{t \in V} \sum_{t_i \in inst(t)} score(t_i, c(t_i); \theta, E)$
5. Estimar $\hat{\theta}, \hat{E} = argmin_{\theta,E} L$
6. Usar el valor E obtenido como matriz de *embedding*.

De esta familia de modelos, el más destacado es Word2Vec, presentado por Google en el año 2013 [61][60]. Se trata de un conjunto redes de neuronas sencillas, de únicamente dos capas (similares a un *autoencoder*), que reciben como entrada corpus de texto y producen un espacio vectorial a partir del mismo. En dicho espacio vectorial, cada palabra perteneciente al corpus de entrenamiento se codifica como un vector de una longitud típicamente entre 100 y 1.000. Las palabras que tienen unas características similares se encuentran cercanas entre sí en el espacio vectorial generado, por lo que se preserva la información semántica.

Una de las características más interesantes de las representaciones generadas mediante Word2Vec es que permiten llevar a cabo operaciones de analogía entre conceptos. Por ejemplo, si se realiza la operación *V(king)- V(man) + V(woman)*, el resultado que devuelve aproximadamente *V(queen)*. Además, este tipo de representación admite también el uso de métricas de similitud.

La arquitectura que se propone es muy sencilla y se compone únicamente de una capa de entrada, una capa de salida y una capa oculta. Al ser un modelo que resuelve un problema de clasificación multiclase, se le añade una función *softmax* [90] a la salida. En este modelo, la codificación de cada palabra en su vector de *embedding* se produce en la capa oculta. La matriz de pesos de la capa oculta tiene una dimensión de *VxN*, donde V es igual al número de *features* a obtener y N es igual al número de palabras del vocabulario. Por tanto, la transpuesta de esta matriz será otra en la que en cada fila tendremos representada una palabra, y en cada columna el valor de cada palabra para cada entrada del vector de *features*, tal y como se representa en la figura 5

Dentro de Word2Vec, existen dos modelos distintos.

- **Modelo basado en bolsa de palabras continua (CBOW)**
  En este tipo de modelos, el objetivo es predecir la probabilidad de aparición de una palabra dado un contexto. El contexto puede estar compuesto tanto por una palabra como por varias. Este tipo de modelo es más rápido de entrenar, y resulta recomendable para conjuntos de datos pequeños.

- **Modelo *skip-gram***
  En este caso, el planteamiento es opuesto al presentado por los modelos *CBOW*. Ahora, dada una palabra, queremos predecir el contexto que la rodea. Este objetivo es más complejo de predecir, por lo que necesita de una mayor cantidad

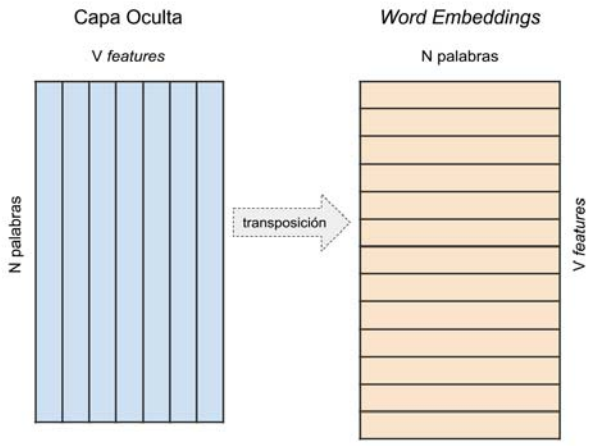


Fig. 5: Representación de la obtención de *word embeddings* por el modelo de Word2Vec.

de datos para entrenar. Sin embargo, los resultados obtenidos por este tipo de modelo son significativamente mejores.

Entrenar este tipo de modelo, además, es muy costoso computacionalmente. Para solucionarlo, se emplea la técnica del muestreo negativo o *negative sampling*, en la cual para un ejemplo de pareja de palabra-contexto positivo, se da un número pequeño de N ejemplos de palabra- contexto negativo. De esta manera, en cada iteración se procesa un número pequeño de datos, y no el conjunto entero, lo que aumenta notablemente la eficiencia computacional.

Otro de los métodos más destacados es FastText [12][44][45], una librería desarrollada por Facebook en el año 2016. Esta librería consta de un módulo aprendizaje de *word embeddings* y otro de clasificación de textos.

Para obtener las representaciones de las palabras, se emplea una red de neuronas *skip-gram*. La particularidad de esta aproximación es que busca explotar la información morfológica de las palabras, para lo que divide las mismas en fragmentos, a los que denomina *n-gramas*, y el modelo lo que aprende es la representación de aquellos n-gramas que forman las palabras que componen el conjunto de entrenamiento. La representación de una palabra será la suma de las representaciones de los n-gramas que la componen. Gracias a esto, se pueden obtener vectores para palabras infrecuentes, ya que es muy posible que la mayoría de sus n-gramas tengan una representación; así como para palabras no pertenecientes al conjunto de entrenamiento empleado. Además este modelo permite la portabilidad entre idiomas.

3.4. Modelos basados en tareas

En estos modelos, a diferencia de los modelos basados en predicciones, se busca aprender unas representaciones que se utilizan únicamente para resolver un problema concreto. Para ello, se emplean generalmente redes de neuronas artificiales, a las

cuales se añade una función objetivo que resuelva el problema deseado y, además, la matriz de *embeddings* E de dimensiones NxV, siendo N el número de palabras del vocabulario y V la longitud deseada de los vectores de *embedding*. Esta matriz se añade a la red como un parámetro más. Al considerarlo de esta manera, cuando el modelo entrene y lleve a cabo la actualización de valores de los parámetros, esta actualización afectará también a la matriz E.

Las representaciones obtenidas por este método carecen de significado semántico general, lo que impide emplear medidas de similitud. Además, ya que estas representaciones están generadas *ad hoc* para resolver un problema concreto, no pueden ser reutilizadas para resolver otros problemas. Uno de los problemas típicos donde se emplean este tipo de modelos es en la compleción de bases de conocimiento, como se estudia en la sección 4.

3.5. Otras aproximaciones

Los modelos anteriores son los más populares y los más utilizados, pero existen otros métodos más novedosos, como el propuesto por Nickel, en el que se propone un modelo de geometría hiperbólica denominado "bola de Poincaré", el cual permite capturar propiedades jerárquicas que no se pueden representar de manera directa en un espacio Euclídeo [70].

Otro trabajo interesante es el realizado por Bian [11], en el cual propone una forma de aprovechamiento de la información morfológica para generar las representaciones, además de realizar una exploración de la influencia de la incorporación de información sintáctica y semántica adicional durante el proceso de entrenamiento del modelo para aumentar la calidad de los vectores.

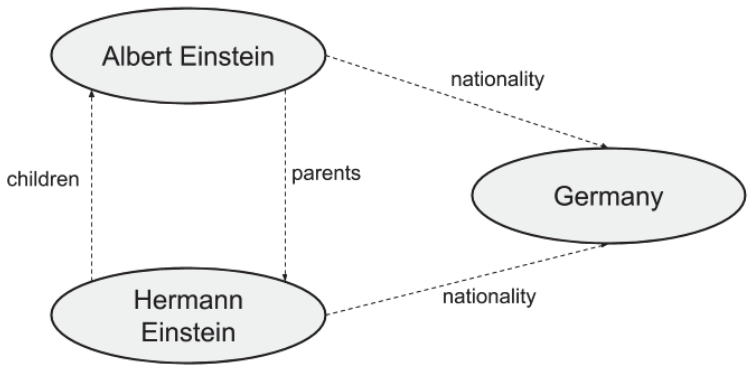
Finalmente, el trabajo de Lai estudia los aspectos críticos que determinan qué compone un buen modelo de *word embedding* [52]. De acuerdo con este trabajo, los tres aspectos más importantes que caracterizan un buen método de *word embedding* son: el modelo, el corpus y los parámetros de entrenamiento.

Además de para representar palabras, las técnicas de *embedding* pueden extenderse a párrafos, documentos [54] y oraciones [46] [87]. En estos casos se suelen emplear modelos basados en predicción, de manera que las representaciones obtenidas son empleadas para llevar a cabo otras tareas, tales como detección de sentimientos o clasificación de textos.

4. BASES DE CONOCIMIENTO

Las bases de conocimiento son un tipo especial de bases de datos destinadas a recolectar, organizar y gestionar conocimiento. Estos sistemas de almacenamiento del conocimiento, junto con los motores de inferencia, constituyen los sistemas basados en el conocimiento. En estos sistemas, se consta de una base de conocimiento que representa hechos conocidos acerca de un dominio concreto, y de un motor de inferencia, que permite razonar acerca de esos hechos, así como emplear reglas u otras formas de razonamiento para detectar inconsistencias o para obtener nuevos hechos. A diferencia de las bases de datos relacionales, la información almacenada no está estructurada, y puede tener una gran complejidad.

Uno de los aspectos más importantes de una base de conocimiento es la calidad de la información que contiene, por lo que debe ser constantemente revisada y actualizada. Además, las bases de conocimiento cuentan habitualmente con un motor de búsqueda, que permite realizar consultas o *querys*. En este tipo de sistemas no se emplea la estructura relacional, sino que se representan una serie de elementos, o entidades, y las relaciones existentes entre los mismos. Debido a esto, una base de conocimiento pueden ser vistas de manera abstracta como un grafo [31], en la cual las entidades corresponden a los nodos y, las aristas a las relaciones entre ellas, por lo que muchas veces son referidas como grafos de conocimiento. De acuerdo a este modelo, los hechos se pueden representar mediante tripletas de la forma $(sujeto, predicado, objeto)$, tal y como se muestra en la figura 6.



sujeto	predicado	objeto
Albert Einstein	nationality	Germany
Hermann Einstein	nationality	Germany
Albert Einstein	parents	Hermann Einstein
Hermann Einstein	children	Albert Einstein

Fig. 6: Ejemplo de grafo de conocimiento y conversión del mismo en tripletas o hechos

4.1. Bases de conocimiento empleadas habitualmente

Las bases de conocimiento recogen un conjunto de hechos asociados a un dominio. Este dominio puede aplicar tanto a un área concreta como ser universal.

WordNet

WordNet [62] [23] es la principal base de conocimiento léxico en inglés, desarrolada en 2005 por la universidad de Princeton. En ella, los nombres, verbos, adjetivos y adverbios se agrupan en conjuntos de acuerdo a la sinonimia entre ellos. Estos conjuntos reciben el nombre de *sysnets*, y cada uno de ellos está asociado a un concepto único. Los *sysnets*, se relacionan entre sí mediante relaciones de tipo semántico-conceptuales y léxicas. Debido a su estructura y al tipo de información que modela, WordNet es una de las herramientas más utilizadas en el área de la lingüística computacional y el procesamiento del lenguaje natural.

A nivel superficial, WordNet puede ser visto como una combinación entre un tesauro y un diccionario, debido a la agrupación que realiza de las palabras de acuerdo a su significado. Sin embargo, esto no es del todo cierto ya que WordNet establece relaciones entre las distintas acepciones que puede tener una palabra, es decir, se lleva a cabo una desambiguación semántica.

En lo referido a la estructura, la principal relación que se considera entre palabras es la sinonimia, de manera que aquellas palabras que denoten el mismo concepto y sean intercambiables en múltiples contextos serán agrupadas en el mismo conjunto o *sysnet*. En total, existen 117.000 *sysnets* interconectados entre sí mediante relaciones conceptuales. Cada *sysnet* contiene tanto una descripción como un ejemplo de uso de los elementos del mismo. Aquellas palabras que presenten polisemia se representan en tantos *sysnets* como significados tengan.

En cuanto a las relaciones, el tipo principal de relaciones entre *sysnets* representadas son aquellas que expresan jerarquía. Esto permite hacer una distinción entre aquellos *sysnets* que representan tipos y aquellos que representan instancias (nombres de personas, países, etc.). Otro tipo de relaciones representadas son las relaciones de parte-conjunto (meronimia y holonimia), y las relaciones de similitud entre conceptos.

Tanto el conjunto de hechos como la ontología empleada están disponibles de manera abierta. La última versión disponible es la 3.0, siendo además la versión final, ya que actualmente esta base de conocimiento no se encuentra en mantenimiento.

Freebase

Freebase [13] es una base de conocimiento colaborativa lanzada en el año 2007 por la compañía americana Metaweb. Esta base de conocimiento busca almacenar conocimiento general, sin restringirse a un dominio concreto, con el objetivo de crear un recurso global que permita tanto a las personas como a las máquinas acceder a la información común de una manera más efectiva. Inicialmente, estaba compuesta tanto de datos obtenidos de manera online de múltiples fuentes (Wikipedia [98],

NNDB [3], MusicBrainz [91]...). como de colaboraciones realizadas por los usuarios, pudiendo ser accedida y utilizada de manera libre.

Tras la compra de Metaweb por parte de Google, Freebase pasa a servir base para el desarrollo del Google Knowledge Graph [31], la base de conocimiento de Google lanzada en el año 2012. Con este modelo, se busca enriquecer y mejorar las respuestas devueltas por el motor de búsqueda Google. Para ello, cuando se realiza una búsqueda de un elemento, no sólo se busca las páginas web asociadas, sino que se extraen también todos los hechos existentes en el grafo asociados a dicho elemento. La información extraída se dispone dentro de una caja que aparece a la derecha de la página de resultados, tal y como se muestra en el ejemplo de la figura 7. Esta base de conocimiento, además de ofrecer una interfaz para realizar búsquedas y consultas sobre la misma, está disponible para su uso y descarga de manera libre.

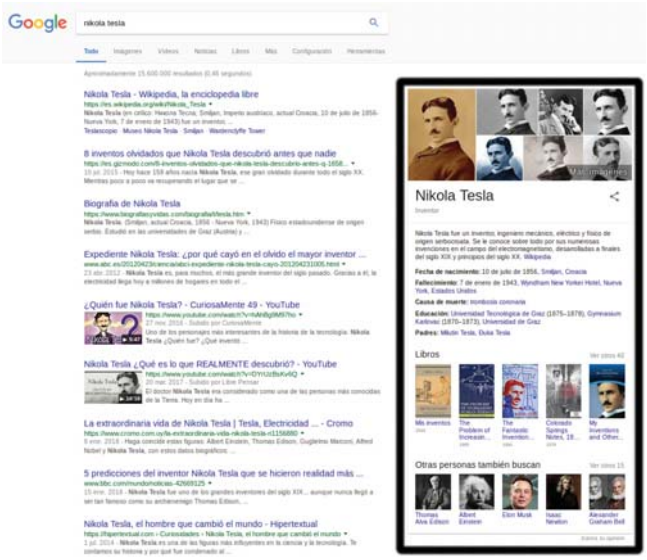


Fig. 7: Resultado de búsqueda de un elemento en Google con la información extraída del Google Knowledge Graph.

Hasta el año 2016, Google lleva a cabo el mantenimiento de Freebase al ser esta la base de conocimiento que sirve como base al Knowledge Graph. En este año, se cesa el mantenimiento de la misma, y el Knowledge Graph pasa a emplear Wikidata [94], una base de conocimiento que recoge la información de Wikipedia [98] transformada en el formato de los hechos propios de este tipo de modelos. Esta base de conocimiento es actualmente una de las más utilizadas, y está disponible de manera abierta.

DBpedia

DBpedia es un proyecto conjunto iniciado en 2007 por la universidad de Berlín, la universidad de Leipzig y la compañía OpenLink Software, cuyo objetivo es extraer información estructurada procedente de múltiples proyectos de Wikimedia, como por ejemplo Wikipedia [98]. La información extraída se estructura en una base de

conocimiento, y es clasificada de acuerdo a una ontología que está disponible de manera abierta. Al emplear proyectos tales como Wikipedia como base, el dominio que abarca DBpedia es universal.

En la actualidad, la versión en inglés de DBpedia recoge más de 4.58 millones de elementos, de los cuales 4.22 millones están clasificados de acuerdo a la ontología. Además, está disponible en 125 idiomas diferentes. El conjunto de hechos también puede ser obtenido de manera libre, aunque también cuenta con un *endpoint* sobre el cual se pueden realizar consultas a la base de conocimiento y recuperar datos de la misma.

Una de las principales ventajas de DBpedia frente a otras bases de conocimiento es la gran extensión de dominios que abarca, además de evolucionar de manera automática cuando se produce una actualización en Wikipedia, por lo que se encuentra en constante mantenimiento y asegura una buena calidad en su información.

4.2. Ontologías

Una ontología es una definición formal de conceptos en un dominio concreto, así como de las propiedades de cada concepto y las relaciones entre ellos. Parten de la rama de la filosofía conocida como metafísica, que busca explicar la existencia mediante la representación de las entidades, las relaciones entre ellas y las propiedades de las mismas. Esta forma de representación es mucho más adecuada para representar el conocimiento que la forma relacional, por lo que en algunas bases de conocimiento, además de recoger las diferentes entidades y las relaciones entre ellas, se incopora una ontología que permite caracterizar dichas entidades. Esto hace que la entidad tenga una información adicional que es ajena a la proporcionada por las relaciones de la misma con otras entidades, dándole así un contexto propio.

Las partes principales de una ontología son:

- **Clases o conceptos**: Conceptos extraídos del dominio a representar. Las clases se organizan de acuerdo a una jerarquía taxonómica (subclase de- superclase de).

- **Ranuras, roles o propiedades**: Conjunto de características o atributos asociados a un concepto. El conjunto de valores que pueden tomar dichos atributos ha de ser definido.

- **Facetas o restricciones de rol**: Restricciones aplicadas sobre las propiedades.

- **Relaciones**: Forma en que las clases se relacionan unas con otras.

Una ontología, junto con un conjunto de individuos o instancias de las clases, constituye una base de conocimiento. En el trabajo *Ontology Development 101* [73], se exponen los principales motivos de por qué desarrollar una ontología, además de proporcionar una guía detallada explicando el proceso de desarrollo a seguir.

No existe una forma única de división de las ontologías en tipos, ya que en función a la propiedad empleada como referencia se pueden obtener diferentes tipos. Atendiendo al dominio al que aplican, distinguimos tres tipos distintos:

Ontologías de dominio

Una ontología de dominio específico representa conceptos asociados a un ámbito concreto, como por ejemplo deporte o política. En este tipo de ontologías, el significado aplicado a un término está restringido por el dominio de la ontología. Por ejemplo, el término *carta* tendrá un significado diferente si se refiere al dominio de la comunicación o al dominio del juego.

El problema principal de este tipo de ontologías es que, debido a su alto nivel de especifidad, cuando es necesario combinar múltiples ontologías de este tipo pueden aparecer conflictos en la denominación de los términos.

Ontologías de alto nivel

Este tipo de ontologías modelan conceptos generales y las relaciones entre ellos, las cuales son comunes a cualquier dominio. Su función principal es servir como soporte para la interoperabilidad semántica entre un gran número de ontologías de dominio, ofreciendo un punto común de partida a todas ellas para formular sus definiciones específicas. Los términos en la ontología de dominio están categorizadas por debajo de los términos de la ontología de alto nivel. Esta herencia de términos se representa mediante la relación *subclase de*

Algunas ontologías de alto nivel estándar son SUMO *Suggested Upper Merged Ontology*[74], UFO *Unified Fundational Ontology* [32] o DOLCE *Descriptive Ontology for Linguistic and Cognitive Engineering* [26].

WordNet [63] ha sido considerada por algunos como una ontología de alto nivel, debido a que contiene información léxica y es utilizada habitualmente herramienta lingüística para el la generación de ontologías de dominio.

Ontologías híbridas

Se trata de una combinación entre una ontología de alto nivel y una ontología de dominio. Un ejemplo de ontología de este tipo es Gellish [78].

4.3. Compleción de Bases de Conocimiento

Considerando la gran cantidad de conocimiento almacenada en las bases de conocimiento, resulta lógico hacer uso del mismo para llevar a cabo otras tareas. Una de las tareas más interesantes que se derivan de las mismas es la búsqueda de sistemas que permitan llevar a cabo razonamientos sobre el conocimiento almacenado en ellas. La compleción de bases de conocimiento, o *Knowledge Base Completion* (KBC), es una de las principales formas de razonamiento que se busca llevar a cabo sobre las bases (o grafos) de conocimiento.

Sabiendo que los hechos en una base de conocimiento se almacenan en forma de tripletas de la forma $(sujeto, predicado, objeto)$, el objetivo de la compleción

de bases de conocimiento es que, dados dos de los tres elementos de la tripleta, el modelo sea capaz de predecir el elemento restante. Por ejemplo, si se tiene la tripleta $(DonaldTrump, presidentof, ?)$, queremos que el modelo sea predecir que el objeto que falta es USA. Dentro de la compleción de bases de conocimiento, existen dos aproximaciones. La primera de ellas consiste en, dado el sujeto o el objeto y el predicado, predecir el elemento restante; en la segunda, dados tanto el sujeto como el objeto, se busca predecir la relación que une ambos. Esta tarea no sólo sirve para evaluar la capacidad de un sistema para razonar sobre una base de conocimiento sino que, además, puede ser utilizada para expandir bases de conocimiento incompletas, mediante la deducción de nuevos hechos a partir de los existentes.

Para la deducción de nuevos hechos, se emplean de manera habitual métodos basados en el *aprendizaje estadístico relacional* (SLR). El aprendizaje estadístico relacional es una subdisciplina del aprendizaje automático que se encarga de trabajar con modelos de dominio que presentan incertidumbre y cuya estructura relacional es compleja. Dentro de los principales objetivos del aprendizaje estadístico relacional están: predicción de nuevas relaciones entre entidades existentes, predicción de propiedades en entidades y agrupamiento de entidades basado en sus patrones de conexión. Dentro de los diferentes tipos de modelos existentes dentro del aprendizaje estadístico relacional, se destacan dos de ellos debido a su escalabilidad. El primer tipo son los modelos basados de factores latentes (*latent feature models*), dentro de los cuales los más populares son los basados en factorización de tensores y en redes de neuronas; y el segundo tipo son los modelos basados en extracción de patrones observados en el grafo [71]. Dentro de este tipo de modelos, los que ofrecen un mejor funcionamiento son aquellos basados en factores latentes, además de ser los más populares.

Un tensor es el resultado de realizar un producto tensorial de N vectores de espacios, el cual representa el resultado de la interacción de N elementos. En el caso de las bases de conocimiento, dado un conjunto de entidades $\mathcal{E} = \{e_1, ..., e_{N_e}\}$, y un conjunto de relaciones $\mathcal{R} = \{r_1, ..., r_{N_r}\}$, se puede representar cada tripleta existente $x_{ijk} = (e_i, r_k, e_j)$ mediante un valor binario $y_{ijk} \in \{0, 1\}$ que representa la existencia o no de dicha tripleta. De acuerdo con esto, todo el espacio de tripletas $\mathcal{E} \times \mathcal{R} \times \mathcal{E}$ puede ser representado mediante un tensor de orden tres $\mathbf{Y} \in \{0, 1\}^{N_e \times N_e \times N_r}$, tal y como se refleja en la figura 8. Cada posible $\mathbf{Y}$ representa un posible mundo. Para derivar un modelo que represente el grafo de conocimiento completo, se busca estimar la distribución conjunta $P(\mathbf{Y})$ a partir de un conjunto $\mathcal{D} \subseteq \mathcal{E} \times \mathcal{R} \times \mathcal{E} \times \{0, 1\}$ de hechos. Dependiendo del criterio seleccionado para generar la distribución, se obtendrá una representación u otra. La distribución probabilística resultante representa todos los $\mathbf{Y}$ posibles, lo que permite predecir la probabilidad de que exista una tripleta T en base al estado del grafo completo.

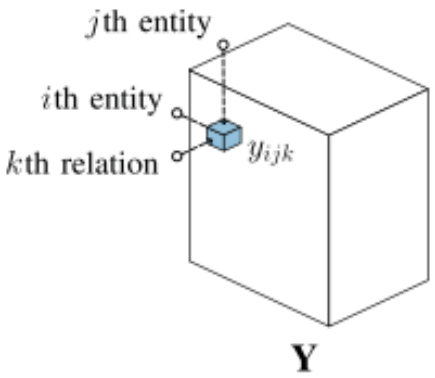


Fig. 8: Ejemplo de tensor $\mathbf{Y}$. Fuente: [71].

4.3.1. Métodos principales de Knowledge Base Completion

RESCAL

Sabiendo que un grafo de conocimiento completo puede ser representado como un tensor, una posible aproximación para llevar a cabo la compleción del grafo es aprovechar las este tipo de representación tensorial, para lo que se utilizan técnicas de factorización tensorial. Factorizar el tensor permite descomponer el grafo, o extraer cierta información del mismo. Dentro de los métodos de compleción basados en factorización tensorial, destacan los siguientes.

RESCAL [72] es uno de los métodos más populares de aprendizaje relacional basado en factorización de tensores. En este modelo, el grafo se representa mediante un tensor de orden tres, tal y como se muestra en la figura 9. Sabiendo que las

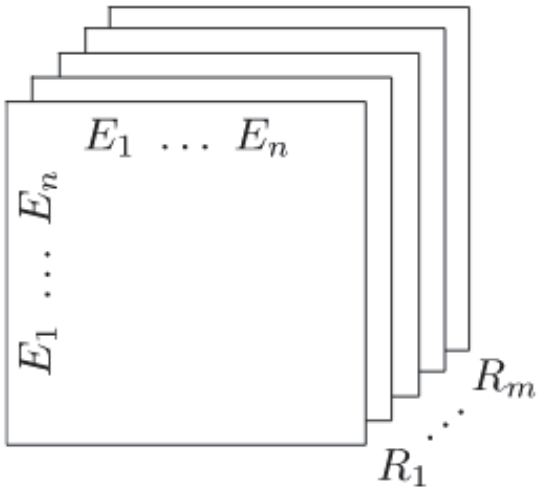


Fig. 9: Modelo de tensor empleado en RESCAL. Fuente: [72].

entidades están conectadas entre sí mediante múltiples relaciones, existirán, por consiguiente, correlaciones directas entre entidades o relaciones, entre las cuales, a su vez, existirán correlaciones. El mecanismo detectar este tipo de correlaciones, así como de explotar la información proporcionada por las relaciones entre entidades es referido como aprendizaje colectivo. Para hacer uso del mismo, se factoriza el tensor del grafo $\mathcal{X}$, tal y como se muestra en 2.

$$\mathcal{X}_k \approx AR_kA^T, \text{para k} = 1,\ldots,\text{m} \tag{2}$$

La matriz A contiene las representaciones de factores latentes de las entidades (la i-ésima fila de la matriz contiene la representación de la i-ésima entidad), mientras que cada matriz R_k modela las interacciones entre los factores latentes de la k-ésima relacion. Estas matrices presentan propiedades muy interesantes. En el caso de A, se cumple que la representación obtenida para cada una de las entidades es única, independientemente de si la entidad actúa como sujeto o como objeto en la relación. Además, dichas cada representación de cada entidad, gracias al aprendizaje colectivo, contiene a su vez la información de todas las relaciones en las que aparece dicha entidad, así como del resto de entidades con las que se relaciona y las correlaciones entre ellas. Tal y como se ilustra en la figura 10, en el ejemplo presentado en [72], las representaciones de factores latentes obtenidas para las entidades *Al* y *Lyndon* serán similares, ya que ambas están relacionadas con el objeto *Party X*. Además, las entidades *Bill* y *John* también tendrán representaciones similares, al estar relacionadas también con el mismo objeto. Gracias a esta propiedad, se cumple que el valor obtenido al realizar la operación $a^T_{Bill}R_{party}a_{PartyX}$ será similar al obtenido tras realizar $a^T_{John}R_{party}a_{PartyX}$, lo que permite predecir exitosamente la relación entre el sujeto *John* y el objeto *PartyX*.

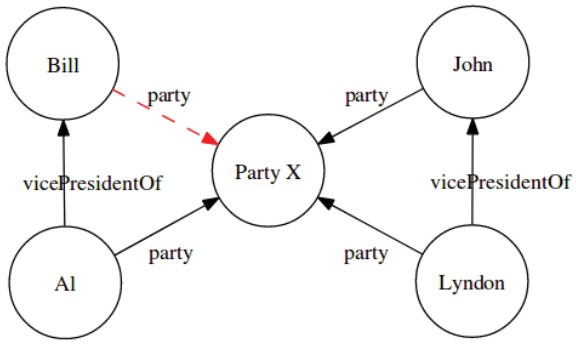


Fig. 10: Ejemplo de predicción de nuevas relaciones entre entidades existentes empleando RESCAL. Fuente: [72].

Neural Tensor Network

La red de neuronas tensorial, o *neural tensor network* (NTN), propuesta por Socher [86], combina la idea de representar el grafo como un tensor con la potencia de cómputo de una red de neuronas, lo que permite predecir nuevos hechos a partir

de únicamente una base de conocimiento, con un alto nivel de acierto. Para ello, cada entidad es representada mediante un vector, los cuales son capaces de capturar hechos acerca de dicha entidad, además de representar cómo de probable es que esa entidad sea parte de una cierta relación. Cada relación se define mediante un conjunto de parámetros de la red que son capaces de relacionar dos entidades de manera explícita.

Este modelo es capaz tanto de aprender representaciones para las entidades de la base de conocimiento como de razonar sobre ellas. Cada relación existente en la base de conocimiento constituye un sub-modelo dentro del modelo principal, de manera que cada relación tiene su propio juego de parámetros. De esta manera, dada una tripleta (e_1, R, e_2), se selecciona el sub-modelo asociado a la relación R, y se le pasan como entrada las entidades e_1 y e_2. El modelo devuelve un valor continuo que será alto si dichas entidades cumplen la relación R, y bajo en caso contrario.Para determinar si una tripleta es correcta o no, se comprueba si el valor devuelto por el modelo para la misma es mayor a un cierto al valor límite (o *threshold*) T_R asociado a la relación R, el cual es obtenido de manera empírica.

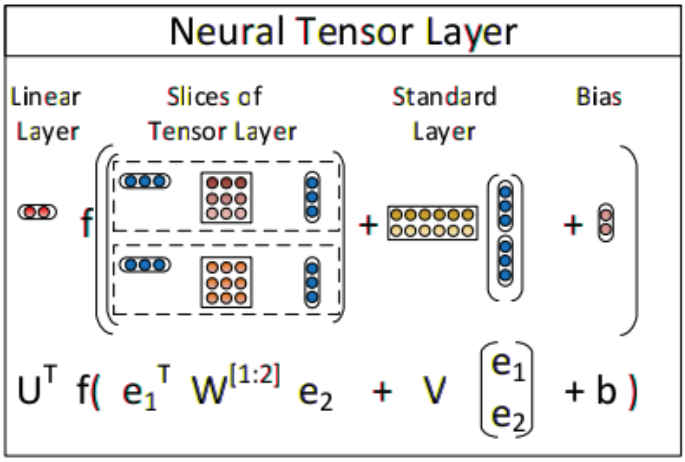


Fig. 11: Visualización de la red de neuronas tensorial. Cada caja representada con una línea discontinua representa un nivel de profundidad (número de rodajas) del tensor. En este caso, el tensor tiene profundidad 2. Fuente: [86].

El modelo tiene la estructura representada en la figura 11. Como se puede apreciar, W representa el tensor del grafo, mientras que el resto de parámetros son las matrices de pesos habituales en las redes alimentadas hacia delante. La operación $e_1^T W^{[1:n]} e_2$ representa el producto bilinear entre dos entidades, de la misma manera que se realiza en RESCAL [72]. La ventaja principal de este modelo es que relaciona las dos entradas de manera multiplicativa, en vez de mediante la concatenación de las mismas, como suele hacerse en las redes de neuronas habitualmente. Además, de manera intuitiva, se puede interpretar que cada rodaja del tensor es responsable de un tipo de pareja de entidades o de una instancia de una relación. De esta manera, el modelo puede llegar a aprender que tanto los animales como los coches pueden

tener partes, aunque pertenezcan a partes muy distintas del espectro semántico. Otra forma de interpretar cada rodaja de el tensor es considerar que cada una de ellas realiza una mediación distinta de la relación entre las entidades de entrada.

Al tratarse de una red de neuronas, es preciso especificar una función de pérdida, que permita ajustar los parámetros del modelo para cumplir el objetivo deseado. El objetivo es que cada tripleta correcta perteneciente al conjunto de entrenamiento $T^{(i)} = (e_1^{(i)}, R^{(i)}, e_2^{(i)})$, debe recibir un valor más alto que una tripleta en la cual una de las entidades ha sido reemplazada de manera aleatoria. Además, existen N_R relaciones, indexadas mediante $R^{(i)}$ para cada posible tripleta, y cada relación tiene asociado su propio conjunto de parámetros. Aquellas tripletas en las que se ha reemplazado una entidad de forma aleatoria son referidas como tripletas corruptas $T_c^{(i)} = (e_1^{(i)}, R^{(i)}, e_c)$. Siendo Ω el conjunto de parámetros de la red, la pérdida a minimizar será la representada en la ecuación 3. El modelo se entrena empleando las técnicas y algoritmos expuestos en la sección 2.

$$J(\Omega) = \sum_{i=c}^{N} \sum_{c=1}^{C} \text{máx}(0, 1 - g^{(T^{(i)})} + g^{(T_c^{(i)})}) + \lambda \parallel \Omega \parallel_2^2 \tag{3}$$

Otra aportación importante realizada en este trabajo es una revisión en la forma de representar las entidades. En este modelo, cada entidad está representada mediante un vector de dimensión fija y, a su vez, una entidad puede estar compuesta de una o más palabras. Generalmente, las representaciones de una entidad pueden ser obtenidas mediante la inicialización aleatoria de la matriz de *embeddigs* E; sin embargo, esto no permite la compartición de la información existente entre las palabras que describen una entidad. Por ello, se propone incorporar un paso previo, en el cual, dado un conjunto de vectores de *embedding* asociadas al vocabulario del conjunto de entidades, cada vector de cada entidad se compone de la media de los vectores de las palabras que la forman. En esta aproximación, la matriz de *embeddigs* E no tendrá una dimensión $E \in R^{d \times N_E}$, siendo N_e el número de entidades, sino que tendrá una dimensión $E \in R^{d \times N_W}$, es decir, existirá un vector por palabra, y en cada iteración del modelo, se recompondrá cada una de las entidades de acuerdo a la ponderación de las palabras que la componen. Ambas aproximaciones son válidas, y en ambos casos, la matriz E es considerada un parámetro más de la red, siendo por tanto modificada en cada iteración que se realiza. La generación de los vectores de las entidades, así como un ejemplo de uso del modelo se refleja en la figura 12.

TransE

TransE [14] es otro de los métodos más populares de compleción de bases de conocimiento. Se trata de un modelo sencillo de entrenar, con un número bajo de parámetros y capaz de escalar a grandes bases de conocimiento. Para ello, las relaciones modeladas en base a una interpretación de las mismas como si fueran operaciones de traslación sobre las representaciones de las entidades. Para aprender las representaciones (o *embeddings*) de las entidades, se emplea un algoritmo basado en la minimización de una función de energía, lo que permite aprender representaciones de las entidades de baja dimensión.

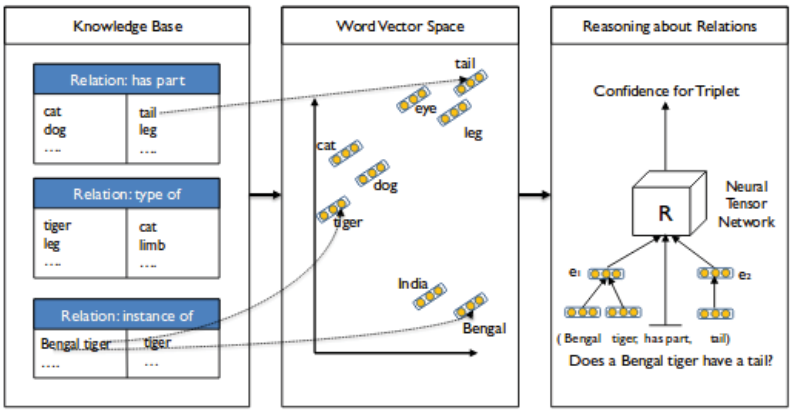


Fig. 12: Visión general del funcionamiento del modelo, así como de la generación de los vectores de entidades. Fuente: [86].

En TransE, las relaciones se representan como traslaciones en el espacio de representaciones, por lo que dada una tripleta $(sujeto, predicado, objeto)$ existente en el grafo, se ha de cumplir que el *embedding* de la entidad objeto sea igual a la suma del *embedding* del sujeto con un vector que depende del predicado que las une. Para generar las representaciones, se emplea un algoritmo sencillo con un conjunto pequeño de parámetros que genera un *embedding* único por cada entidad y por cada relación. Estas representaciones permiten llevar a cabo un proceso de razonamiento análogo al empleado por RESCAL [72], lo que permite detectar nuevas relaciones entre entidades.

TransR

Aunque TransE [14] obtiene unas representaciones válidas mediante la consideración de la relación como una operación de traslación entre las representaciones de las entidades, en este modelo se considera que tanto las entidades como las relaciones pertenecen al mismo espacio semántico. Una entidad puede tener múltiples aspectos, y una relación puede focalizarse también en diferentes aspectos de las entidades, por lo que emplear un espacio común para representar ambos elementos puede resultar insuficiente.

Para resolver este problema, TransR [57] propone un método de *embedding* en el cual se emplean dos espacios independientes para representar las relaciones y las entidades. Al igual que en el caso de TransE [14], se plantean las relaciones como operaciones de traslación. Para permitir esta traslación, considerando que las entidades y las relaciones están en espacios distintos, se asocia a cada relación r una matriz M_r, que proyecta las representaciones del espacio de entidades al espacio de relaciones, tal y como se muestra en la figura 13. Una vez obtenidas las proyecciones de las entidades en el espacio de relaciones, se puede llevar a cabo la traslación asociada a la relación y, consecuentemente, predecir nuevas relaciones entre entidades.

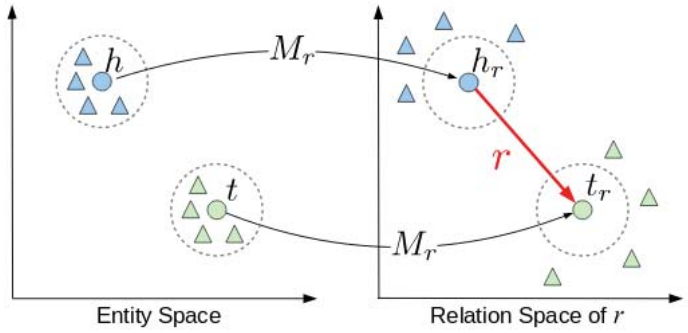


Fig. 13: Diagrama explicativo del modelo TransR. Fuente: [57].

4.3.2. Otras aproximaciones

Además de los métodos mencionados anteriormente, existen múltiples trabajos destacables dentro del área de la compleción de bases de conocimiento. Algunos de ellos emplean alguno de los anteriores como base, estudiando sus limitaciones y proponiendo mejoras y soluciones para las mismas. Dentro de los trabajos que se centran en la *Neural Tensor Network* [86], existen algunos que estudian las características de la misma y buscan mejorar los resultados mediante la optimización de los parámetros de entrenamiento, como el propuesto por Huang et al. [43], y otros trabajos que, al igual que el presente, buscan comprobar si el modelo NTN es capaz de funcionar para tripletas en las que una de las entidades no ha sido procesada durante el proceso de entrenamiento (Cheng et al. [18]).

Otros autores se centran en proponer soluciones para el problema de la predicción de nuevos hechos. Una posible aproximación es la propuesta por West et al. [97], en la cual se presenta un sistema basado en pregunta-respuesta para obtener la información que falta. En este trabajo, lo que se busca es un modelo que aprenda qué consultas web ha de realizar para obtener una información determinada, para, a continuación, filtrar esta información para convertirla en hechos que sean válidos para la base de conocimiento. Otros trabajos, como el propuesto por Gardner y Mitchell [27], emplean extracción de patrones observados en subgrafos para generar las matrices de características del grafo; mientras que trabajos como el propuesto por Wang et al. [96], combina un método basado en reglas con la factorización de tensores.

Finalmente, existe otra corriente de trabajos centrados en enriquecer las representaciones, o *embeddings*. Trabajos como el propuesto por el grupo de investigación de Microsoft [93], en el cual proponen un modelo de *embedding* para bases de conocimiento capaz de capturar la estructura composicional de las relaciones textuales, optimizando así de manera conjunta las representaciones de las entidades, la base de conocimiento y las relaciones textuales. Otros trabajos, como el propuesto por Bordes et al. [15], proponen un modelo basado en una red de neuronas artificial que permite obtener representaciones simbólicas (*embeddings*) de los elementos de cual-

Relation	**Percentage Unknown**	
	All 3M	*Top 100K*
PROFESSION	68 %	24 %
PLACE OF BIRTH	71 %	13 %
NATIONALITY	75 %	21 %
EDUCATION	91 %	63 %
SPOUSES	92 %	68 %
PARENTS	94 %	77 %
CHILDREN	94 %	80 %
SIBLINGS	96 %	83 %
ETHNICITY	99 %	86 %

Tab. 1: Porcentaje de incompletitud para algunas relaciones asociadas a entidades de tipo *PERSONA* en la base de hechos Freebase. *A la izquierda:* Porcentaje de incompletitud para todas las entidades *PERSONA* existentes en la base de conocimiento. *A la derecha:* Porcentaje de incompletitud para las 100K entidades más frecuentes del tipo *PERSONA* existentes en la base de conocimiento. Fuente: [97].

quier base de conocimiento, independientemente de las características de la misma. Por último, Yang et al. [99] proponen un método capaz de aprender representaciones ricas en información semántica para los elementos de la base de conocimiento para, a partir de ellas, obtener un conjunto de reglas que permita llevar a cabo el razonamiento.

4.4. Limitaciones y problemas encontrados

A pesar de los buenos resultados que obtienen los métodos anteriores, todos ellos presentan una importante limitación: no permiten la introducción de nuevas entidades al grafo de conocimiento. En todos los trabajos estudiados, las representaciones, tanto de las entidades, como de las relaciones, como del grafo, son obtenidas a partir de un conjunto de dimensión fija, de manera que sólo aquellos elementos que se encuentran en el grafo inicialmente obtendrán una representación. Además, en el caso de las representaciones de las entidades, se trata de representaciones que no capturan la información global de la misma, sino que reflejan únicamente la información de las relaciones de la misma otras entidades. En todos los trabajos, las representaciones se generan *ad hoc* para el método particular a utilizar, por lo que una misma entidad, dependiendo de si se emplea un método u otro, puede tener dos representaciones muy distintas. Debido a esto, la introducción de nuevos hechos a la base de conocimiento ha de realizarse manualmente, lo que provoca que exista una gran incompletitud en ellas. En la tabla 1 se muestra el porcentaje de incompletitud para ciertas relaciones de la base de hechos Freebase.

Otro de los principales problemas que presentan estos modelos es su falta de escalabilidad, ya que no sólo los hechos deben ser introducidos manualmente, sino

que para que este hecho sea considerado por el modelo predictivo, éste debe ser reentrenado desde cero para incorporar esta información. Esto supone, además, un sobreesfuerzo computacional.

5. PLANTEAMIENTO DEL PROBLEMA

En este trabajo, se plantea una aproximación al problema de la predicción de relaciones entre entidades dentro del ámbito de la compleción de bases de conocimiento, cumpliendo para ello los objetivos planteados en la sección 1.3.

En la mayoría de los métodos estudiados en la sección 4, la manera de caracterizar una entidad y, consecuentemente, predecir las relaciones asociadas a la misma, está basada en la explotación de las correlaciones extraídas de las relaciones de la entidad con otras entidades, no en el contexto general de la misma. Por tanto, a la hora de predecir si dos entidades se relacionan entre sí, esta decisión está basada únicamente en esta información que, si bien ofrece una representación de la entidad válida para dicha tarea, está limitada al contexto del conjunto de hechos de la base de conocimiento. Además, debido a que las relaciones se establecen de manera directa entre las entidades, no existe una restricción explícita en el tipo de entidades que pueden aparecer como sujeto y como objeto de una relación.

Por el contrario, en las ontologías, este tipo de restricciones sí aparecen representadas de manera explícita. En este ámbito, aunque la relación se establece directamente entre dos entidades, o instancias, existe una restricción que especifica el tipo de las entidades que pueden aparecer como sujeto y objeto de una relación determinada. De esta manera, las instancias de una clase heredan el conjunto de restricciones que afectan a la misma, lo que además de aportar consistencia a la hora de establecer las relaciones, aporta una información contextual a las instancias.

Considerando esto, se podría interpretar que los métodos de compleción de bases de conocimiento llevan a cabo un razonamiento de bajo nivel, directamente sobre las entidades, sin inferir restricciones acerca de las mismas. Por el contrario, la representación de relaciones en las ontologías se puede considerar de alto nivel, debido a que se da entre clases, y no entre instancias. En la figura 14, se representa gráficamente la diferencia entre la forma de establecer las relaciones en ambos tipos. A la izquierda se representa la relación entre instancias de las clases *PERSONA* y *PAÍS*, donde se establece la relación directa entre las instancias en la parte inferior y, en la parte superior, se muestra la restricción asociada a la relación *nacionalidad*; a la derecha, la relación establecida de manera directa entre entidades en una base de conocimiento.

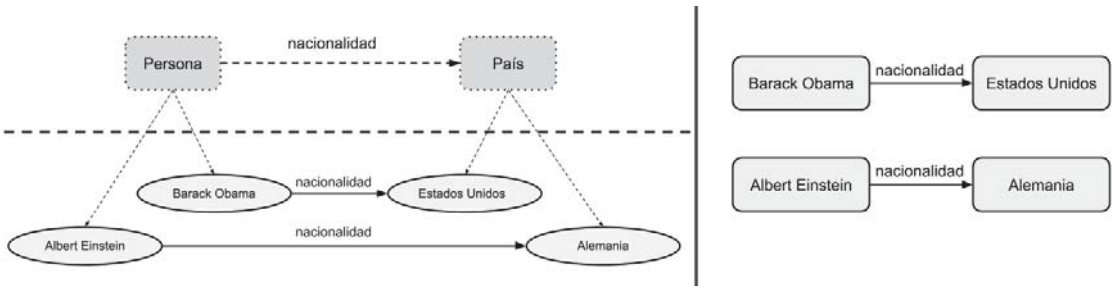


Fig. 14: Comparación entre las relaciones en una ontología *vs* en una base de conocimiento.

Ambas aproximaciones no son excluyentes, sino que pueden ser complementarias entre sí, enriqueciendo así la contextualización de las entidades y mejorando, potencialmente, la capacidad de predicción de relaciones entre ellas. En este trabajo, se introduce información ontológica explícita a un modelo de compleción basado en factores latentes, a fin de estudiar si esto permite, en efecto, mejorar la capacidad de inferencia del mismo. Adicionalmente, se estudia si esta información permite contextualizar entidades desconocidas, permitiendo razonar sobre ellas de la misma manera que sobre las entidades existentes.

5.1. Dificultades encontradas

Llevar esta idea a cabo plantea ciertas dificultades. La primera de ellas es determinar qué modelo emplear. En la sección 4 se presentan los modelos más populares, y los que habitualmente obtienen mejores resultados. Sin embargo, no todos son aptos para la inclusión de información ontológica. Por ejemplo, en el caso de RESCAL [72], el modelo se construye de manera automática a partir del grafo, por lo que si se deseara introducir información ontológica en el modelo, sería necesario convertirla en hechos para ser incorporada a la base de conocimiento, por lo que se perderían las propiedades de la misma. Además, es deseable que el modelo que se vaya a utilizar esté disponible de manera abierta, así como los datos utilizados por los autores para el entrenamiento y la evaluación.

Otro punto importante de este trabajo es determinar qué se va a utilizar como información ontológica. Dependiendo de la base de conocimiento que se emplee, es necesaria una ontología u otra, debido a que, al igual que las bases de conocimiento, están asociadas a dominios concretos. Por tanto, se plantean dos opciones, o bien emplear una ontología de dominio universal que permita la portabilidad de información entre distintas bases de conocimiento; o emplear una ontología de dominio, pero que permita aportar un contexto más rico a las entidades, y unas restricciones más específicas.

Adicionalmente, hay que determinar cómo se va a incorporar esta información al modelo. Convertir la ontología en hechos e incorporarlos a la base de conocimiento supone que se pierde la componente jerárquica, la cual es muy relevante. Por tanto, se precisa una forma de aprovechar esta información e integrarla al modelo de manera que se preserve su integridad y sus características.

Finalmente, hay que determinar cómo codificar esta información ontológica, ya que en los modelos estudiados, sólo las entidades existentes en el grafo de conocimiento tienen una representación vectorial asociada. La ontología, al ser externa al grafo, no cuenta con una codificación, por lo que es necesario plantear cómo codificar esta información para poder integrarla adecuadamente al modelo.

6. SOLUCIÓN PROPUESTA

6.1. Planteamiento general de la solución

El objetivo principal de este trabajo es demostrar que, dado un modelo de compleción de bases de conocimiento, la incorporación de información ontológica en el mismo permite mejorar su capacidad de inferencia. Como se expone en la sección 5, se parte de que la hipótesis de la combinación de la información ontológica junto a la información contextual deducida de los hechos existentes en la base de conocimiento, permite caracterizar mejor las entidades y, consecuentemente, facilitar el proceso de razonamiento.

El motivo principal para emplear información ontológica es que, mientras que la información extraída de las correlaciones existentes la base de hechos es volátil y dependiente del conjunto de datos utilizado, la información proporcionada por la ontología es constante e independiente. Por ello, se espera que el modelo lleve a cabo una explotación de la misma, llegando a inferir para las relaciones de la base de conocimiento unas restricciones de relación similares a las que existen de manera explícita en la ontología. En este trabajo, se considera información ontológica el árbol de clases que representa a una entidad. Además, no se consideran las propiedades de las clases a fin de facilitar la generalización.

En la figura 15 se muestra visualmente el tipo de restricciones deducidas por el modelo al introducir información ontológica sobre las entidades. En la parte superior de la imagen, se muestra la restricción abstracta que determina la clase de las entidades que aparecen tanto de sujeto como de objeto. En la parte inferior, las representaciones de las entidades a las que se les ha añadido una etiqueta con el valor de la clase a la que pertenecen.

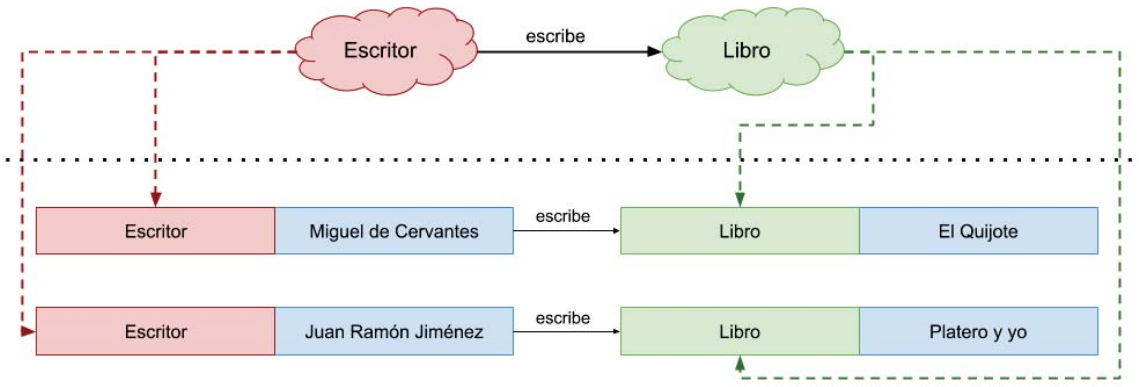


Fig. 15: Extracción de patrones de una relación combinando información ontológica con la información extraída de la base de hechos.

6.2. Consideraciones previas

6.2.1. Selección del modelo predictivo

Uno de los factores más determinantes de este trabajo es elegir correctamente el modelo predictivo a emplear. Se necesita un modelo que tenga una buena capacidad predictiva, y que ésta sea demostrable. Además, ha de permitir la introducción de información ontológica de una manera relativamente sencilla, y que no implique una alteración en su diseño. También, el modelo ha de ser computacionalmente viable dados los recursos disponibles.

Tras estudiar todos los modelos disponibles, y de acuerdo a los requisitos planteados, se decide utilizar como modelo predictivo la red de neuronas tensorial (*Neural Tensor Network*) [86]. Dentro de los motivos por los que se decide emplear este modelo, uno de los más relevantes es la disponibilidad del mismo. El código original, así como los conjuntos de datos empleados por los autores, están publicados de manera abierta, haciendo que éste sea reutilizable. Otra ventaja es que, al gozar de una gran popularidad, existen varias implementaciones del modelo empleando diferentes librerías y lenguajes, lo que aporta un amplio marco de referencias [85][5].

Finalmente, la característica más interesante de este modelo, y lo que le hace más apropiado para este problema, es que los *embeddings* de las entidades no han de ser generados directamente por el modelo, sino que pueden emplearse representaciones externas. De acuerdo con lo expuesto en la subsección 4.3.1, la matriz de *embeddings* de las entidades, E, se inicializa de manera externa al modelo y se incorpora como un parámetro más de la red. Poder decidir cómo y qué valores han de tener inicialmente las representaciones, así como poder decidir si son o no modificadas durante el proceso de entrenamiento, hace de este modelo una buena opción para la resolución del problema.

6.2.2. Introducción de información ontológica

Tal y como se expone al inicio de la sección, en este trabajo la información ontológica de una entidad es la taxonomía de clases de la misma, es decir, el conjunto de clases que dan lugar a la entidad, recorriendo recursivamente y de manera ascendente el árbol de clases. Por ejemplo, dada la entidad *Miguel de Cervantes*, y la jerarquía de clases de DBpedia [1], *Miguel de Cervantes* es una instancia de la clase *ESCRITOR*. A su vez, la clase *ESCRITOR* es una subclase de la clase *PERSONA*, la cual es subclase de la clase *AGENTE*. Por tanto, la información ontológica de la entidad *Miguel de Cervantes* estaría compuesta por las clases *ESCRITOR, PERSONA* y *AGENTE*. El conjunto de clases obtenidas para una misma entidad varía dependiendo de la ontología empleada.

Obtenidas las clases con las que se relaciona una entidad, es necesario decidir cómo se van a codificar las clases, y cómo se van a incorporar al entrenamiento. Dadas las características del modelo, se determina que la forma óptima de codificar esta información es de manera vectorial. Considerando que las representaciones de las entidades se recogen en una matriz que actúa como diccionario, en la que cada entidad lleva asociada su representación, se decide emplear este mismo criterio para

representar las clases generando, de manera análoga, una matriz en la que por cada cada clase de la ontología exista una representación vectorial única. Mantener ambas matrices de representaciones separadas permite que se pueda entrenar, o no, la matriz de representaciones de las entidades, lo que permite obtener representaciones con información contextual extraída de la base de conocimiento, pero manteniendo constante la matriz de representaciones de las clases.

Finalmente, falta determinar cómo se va a llevar a cabo la introducción de esta información en el entrenamiento. Para preservar la arquitectura del modelo, se decide incorporar esta información en los datos de entrada. Una posible aproximación es concatenar los vectores de las clases junto al vector de la entidad, formando un único vector. Sin embargo, esta solución no es válida, ya que no todas las entidades tienen el mismo número de clases asociadas, por lo que los vectores obtenidos tienen dimensiones dispares. Además, en esta forma de representación, la entidad en sí pierde relevancia, al representar una parte muy pequeña del vector final. Por tanto, para mantener la consistencia en las representaciones, así como la relevancia de la información particular de cada entidad, se decide codificar la información ontológica de todas las entidades en un único vector, resultante de realizar la media aritmética de los vectores de las clases de la entidad. Este vector obtenido, que tiene dimensión fija e igual para todas las entidades, es concatenado a la izquierda del vector de la entidad. El vector resultante de la concatenación es el que se emplea como entrada, de manera que durante el entrenamiento, el modelo pueda emplear esta información para la extracción de patrones. En la figura 16 se muestra el proceso completo de generación de los vectores de entrada.

6.2.3. Bases de conocimiento empleadas

Determinado ya el modelo y la metodología de introducción de la información ontológica, falta determinar tanto qué ontologías se van a emplear, como las bases de conocimiento sobre las que se va a trabajar.

Ya que se va a emplear como modelo la red de neuronas tensorial [86], la cual ha sido entrenada y evaluada tanto con *WordNet* como con *Freebase*, resulta coherente emplear también estas bases de conocimiento en el presente trabajo. Además, al igual que el código original del modelo, los autores de la red de neuronas tensorial ofrecen de manera libre los datos empleados en el trabajo original extraídos de ambas bases de conocimiento, así como la matriz de *embeddings* de palabras E inicial asociada al vocabulario de ambas.

El conjunto de hechos extraídos para estas bases de conocimiento están particionados en los tres conjuntos habituales empleados en el aprendizaje automático: entrenamiento, validación y evaluación. El conjunto de entrenamiento está compuesto únicamente por ejemplos correctos, ya que la generación de ejemplos incorrectos se lleva a cabo *ad hoc* durante el entrenamiento. En el caso de los conjuntos de validación y test, se componen de tripletas etiquetadas como correctas (1) o incorrectas (-1). El objetivo del modelo es, por tanto, determinar la veracidad, o no, de una tripleta o hecho desconocido. Los conjuntos de validación y evaluación contienen, por cada tripleta correcta (e_1, R, e_2), una tripleta incorrecta (e_1, R, e_c) en la que la

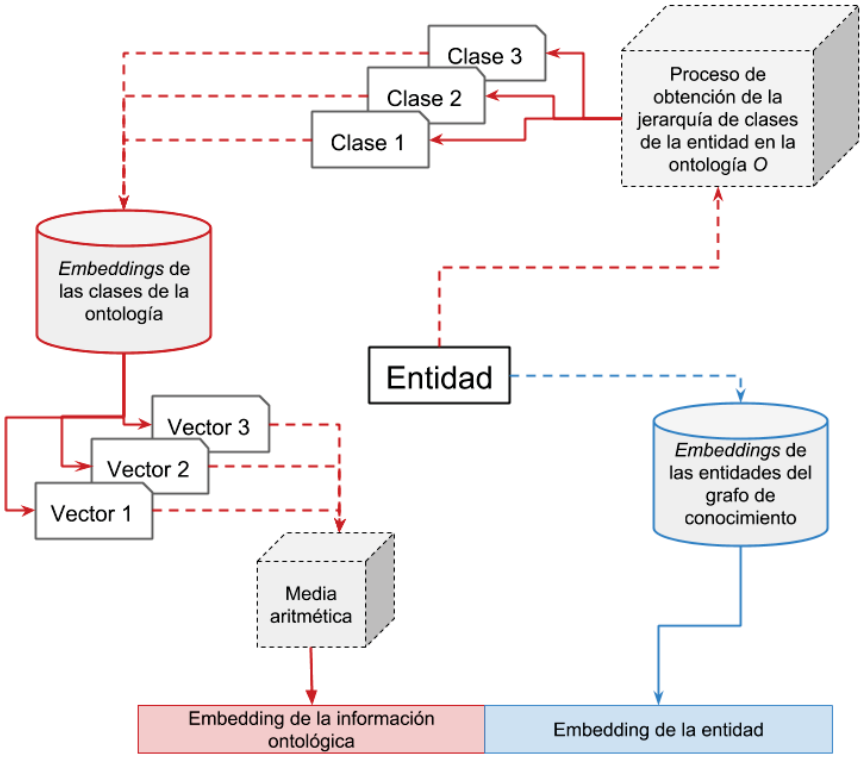


Fig. 16: Proceso de obtención de los vectores de entrada del modelo incorporando información ontológica.

entidad objeto ha sido reemplazada aleatoriamente por otra extraída del conjunto total de entidades. De esta manera existe el mismo número de tripletas correctas que de tripletas incorrectas en ambos conjuntos.

Del total de relaciones modeladas en *WordNet*, el conjunto empleado por Socher et al. [86] recoge hechos asociados a once de ellas: *has_instance, type_of, member_meronym, member_holonym, part_of, has_part, subordinate_instance_of, domain_region, sysnet_domain_topic, similar_to* y *domain_topic*. Del total de relaciones representadas en *Freebase*, Socher et al. [86] recogen hechos de aquellas relaciones pertenecientes al dominio *PERSONA*. Este subconjunto está compuesto por un total de trece relaciones: *gender, nationality, profession, place_of_death, place_of_birth, location, institution, cause_of_death, religion, parents, children, ethnicity* y *spouse*. En este conjunto, hay relaciones que son más triviales de predecir que otras, como el género, que sólo puede tomar dos valores, o la religión, ya que el conjunto de valores posibles que puede tomar el objeto en la tripleta $(e_1, religion, ?)$ es relativamente pequeño.

La relación de la cantidad de hechos que se recogen en cada uno de los conjuntos (entrenamiento, validación y evaluación) por base de conocimiento, así como el número de entidades y el número de relaciones representadas, se recoge en la tabla 2.

Base de conocimiento	#Relaciones	#Entidades	#Entrenamiento	#Validación	#Evaluación
Wordnet	11	38.696	112.581	5.218	21.088
Freebase	13	75.043	316.232	11.816	47.466

Tab. 2: Relación del número de relaciones y de entidades representadas para *Wordnet* y *Freebase*. También se indica el número de tripletas de entrenamiento, validación y test por cada conjunto. Fuente: [86].

6.2.4. Fuentes de conocimiento ontológico empleadas

En cuanto la generación de información ontológica, a lo largo de este trabajo se emplean dos fuentes distintas. Una de las bases empleadas como fuente de información ontológica es *Wordnet*. Tal y como se expone en la sección 4.1, esta base de conocimiento tiene un carácter pseudo-ontológico, ya que una gran cantidad de las relaciones modelizadas en ella representan una jerarquía entre entidades, o clases. De la serie de relaciones representadas, hay tres que son análogas a las existentes en una ontología habitualmente: *has_instance, subordinate_instance_of* y *type_of*. Por ejemplo, la tripleta $(novelist, has\_instance, Goethe)$, nos indicaría que la entidad *Goethe* es una instancia de la clase *NOVELIST*, y a su vez la tripleta $(novelist, type\_of, author)$, nos indicaría que la clase *NOVELIST* es una subclase de la clase *AUTOR*. Este encadenamiento de hechos permiten obtener la jerarquía de clases que dan lugar a la entidad *Goethe*, que es el tipo de información ontológica que queremos representar. Además, ya que los vectores de las entidades de *WordNet* son conocidos de manera previa, esto permite componer de manera sencilla los vectores de las clases y generar la matriz donde se almacena cada clase con su correspondiente representación.

Sin embargo, la información ontológica que se puede extraer de *WordNet* sólo tiene aplicación dentro de la propia base de conocimiento, ya que las entidades existentes en ella no son las mismas que las existentes en *Freebase*. Por tanto, para obtener la información ontológica correspondiente a las entidades de *Freebase*, es necesario emplear otra base de conocimiento ontológico.

En este caso, se decide emplear *DBpedia* como fuente de conocimiento ontológico para las entidades de *Freebase*, debido a que tiene dominio universal, el cual incluye la clase *PERSONA* y, consecuentemente, un gran conjunto de instancias de la misma. Además, incluye también instancias de otros muchos dominios, lo que garantiza que, aunque puede haber instancias que sólo existan en *Freebase*, hay un número elevado de entidades que son comunes a ambas. De esta manera, podemos aprovechar los conjuntos de datos de *Freebase* ya existentes, permitiendo además establecer comparativas entre los valores obtenidos por el modelo al emplear, o no, información ontológica.

6.3. Fase 1: Implementación del modelo predictivo

Determinado el modelo a utilizar, es necesario llevar a cabo la implementación del mismo. El código original del modelo data del año 2013, y está pensado pa-

ra funcionar en un entorno MATLAB. Sin embargo, recientemente han aparecido múltiples librerías dedicadas a dar soporte a algoritmos de aprendizaje automático, como Tensorflow o Pytorch, ya mencionadas en la sección 2.5. Emplear este tipo de librerías simplifica en gran medida el desarrollo ya que, por ejemplo, mientras que en el caso de MATLAB es necesario calcular el gradiente del error y las actualizaciones de los pesos de manera manual, tanto Tensorflow como Pytorch cuentan con funciones propias que llevan a cabo automáticamente este proceso.

De las dos librerías mencionadas, se decide emplear Pytorch, al ser, a criterio personal, más intuitiva y más sencilla de utilizar, pero manteniendo todas las facilidades y ventajas ofrecidas por Tensorflow. Aunque Pytorch permite de manera directa crear arquitecturas de redes de neuronas artificiales, al no ser este modelo una red convencional, no se pueden aprovechar estas facilidades de diseño, por lo que será necesario implementar el modelo. También es necesario codificar la función de pérdida presentada en el trabajo original, de manera que tanto el modelo como el proceso de aprendizaje sean idénticos a los originales.

6.3.1. Diseño e inicialización

De acuerdo con las normas de diseño de Pytorch, si se desea crear un nuevo modelo desde cero, éste ha de ser una instancia de la clase *Module*. En este tipo de objetos es necesario indicar, al menos, cómo se realiza la inicialización, y la serie de operaciones que se han de realizar para, dada una entrada, devolver el valor de salida.

En cuanto al la inicialización del modelo, podemos dividir el proceso en dos partes. En la primera parte, necesitamos determinar el valor de los hiperparámetros del modelo, los cuales son pasados mediante el constructor, pudiendo ser por tanto especificados por el usuario.

Dichos hiperparámetros son:

- ***k:*** Número de rodajas del tensor.
- ***d:*** Dimensión de los vectores de *embedding* de las entidades.
- ***r:*** Número de relaciones a entrenar
- ***embedding_matrix:*** Valor inicial de la matriz de *embeddings* de las entidades.
- ***train_embeddings:*** Valor booleano que indica si la matriz de *embeddings* de las entidades va a ser considerada como parámetro y, consecuentemente, modificada en tiempo de entrenamiento, o si sus valores van a permanecer fijos durante el proceso.

Cuando queramos incorporar información ontológica al modelo, será necesario especificar, además, el valor de los siguientes hiperparámetros:

- ***ont_info:*** Valor booleano que indica si se va a emplear, o no información ontológica.

Parámetro	Dimensiones	Método de inicialización
$\boldsymbol{W}$	$(d \times d \times k \times r)$	*Algoritmo Xavier*
$\boldsymbol{V}$	$(k \times 2d \times r)$	*Algoritmo Xavier*
$\boldsymbol{b}$	$(k \times 1 \times r)$	*Todos los valores a cero*
$\boldsymbol{U}$	$(1 \times k \times r)$	*Todos los valores a uno*

Tab. 3: Especificación de los parámetros de la red de neuronas tensorial.

- ***ont_matrix:*** Matriz donde se recogen los *embeddings* de las clases de la ontología.

Una vez especificados los hiperparámetros del modelo, se lleva a cabo la segunda parte, que consiste en inicializar los parámetros internos del modelo. Tal y como se especifica en la sección 4.3.1, el modelo cuenta con cuatro parámetros: W, V, b y U, a los que hay que añadir la matriz de *embeddings* E. W es un parámetro que representa el tensor asociado la base de conocimiento, mientras que V, b y U son parámetros habituales de una red de neuronas: V y U son dos matrices de pesos, y b es el valor de la desviación o *bias*. Cada parámetro, además, cuenta con una dimensión adicional igual al número de relaciones existentes, lo que permite que cada relación tenga su propio conjunto de parámetros, pero permitiendo explotar las correlaciones existentes entre las relaciones. La especificación de los parámetros con sus valores de inicialización se recoge en la tabla 3.

6.3.2. Especificación del conjunto de operaciones realizadas

Una vez diseñado e inicializado el modelo, es necesario especificar el conjunto de operaciones que lleva a cabo para convertir la entrada en un valor de salida. En Pytorch, esta serie de operaciones deben ser especificadas mediante una función denominada función *forward*. En esta función, se opera cada tripleta (e_1, R, e_2) de la forma especificada en la ecuación 4, obteniendo la puntuación asociada a la misma.

$$score(e_1, R, e_2) = U_R^T f(e_1^T W_R^{[1:k]} e_2 + V_R \begin{bmatrix} e_1 \\ e_2 \end{bmatrix} + b_R) \tag{4}$$

Ya que existe un conjunto de parámetros por cada relación, deberemos entrenar cada uno de ellos de forma semi-independiente. En la imagen 17, se muestra el desglose de una iteración de entrenamiento realizada por el modelo. Mediante el hiperparámetro *batch_size*, el usuario indica cuántas tripletas del conjunto total de entrenamiento son tomadas en cada iteración. Como se observa, el conjunto seleccionado de ejemplos de entrenamiento está dividido en subconjuntos, en el cual se agrupan las tripletas seleccionadas de acuerdo al valor de su relación. Estos subconjuntos, a su vez, se componen por un número N de ejemplos correctos, para cada uno de los cuáles se genera, mediante reemplazo aleatorio, un número *N_corruptas* de tripletas incorrectas. El conjunto de las tripletas correctas e incorrectas seleccionadas para una relación componen el *batch* de la misma para esa iteración. Una vez establecido el conjunto de datos correspondiente a cada relación, se hace pasar cada

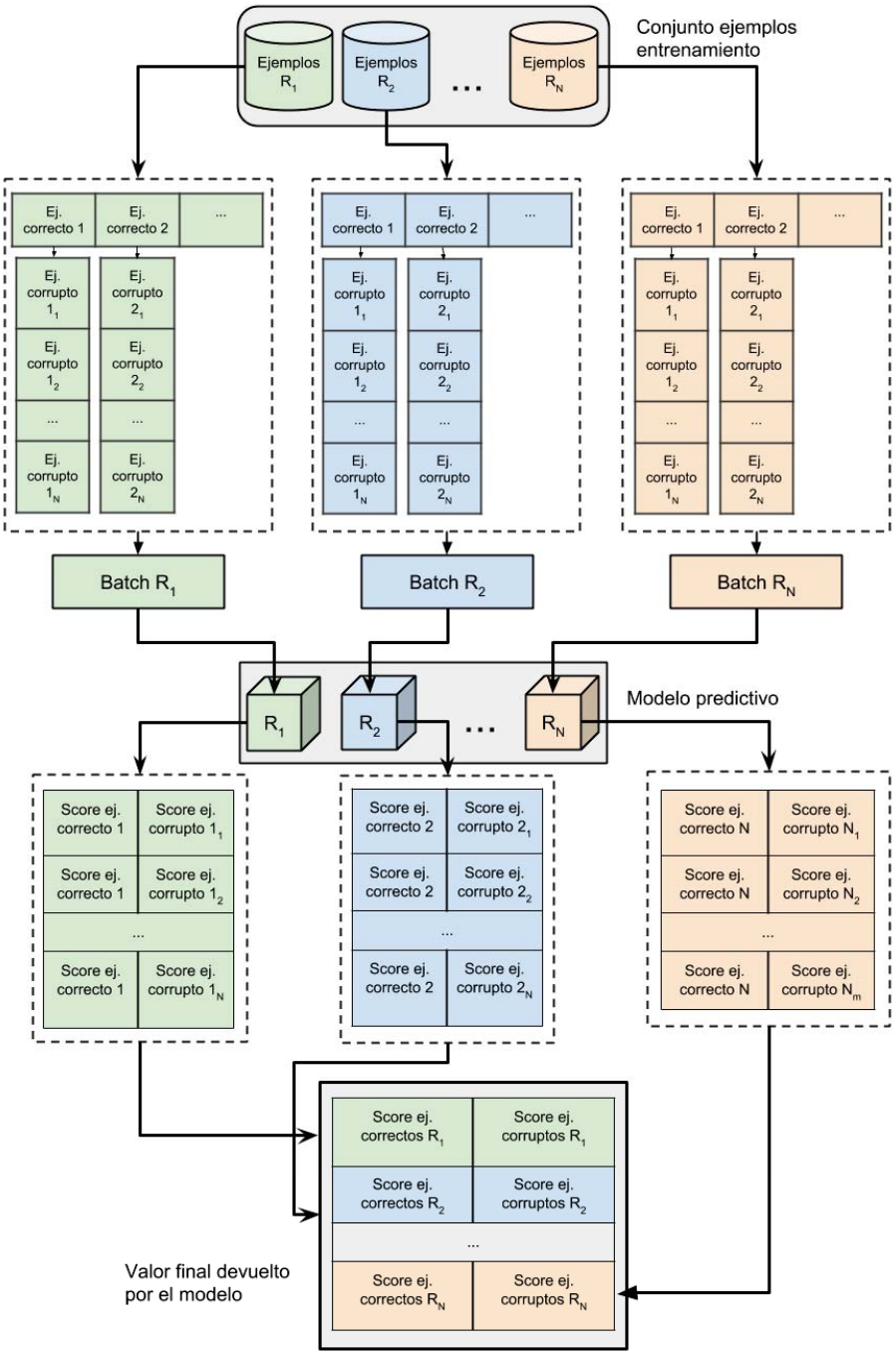


Fig. 17: Representación de la iteración de entrenamiento realizada por la red de neuronas tensorial.

una de las parejas *tripleta correcta-tripleta corrupta* por el modelo correspondiente, obteniendo de igual manera una tupla *(puntuación tripleta correcta, puntuación tripleta incorrecta)* como valor de salida. Este proceso se repite para todas las parejas del *batch* de la relación, y, de igual manera, para todas las relaciones del modelo. Una vez se han obtenido todas las puntuaciones de todas las tripletas correctas e incorrectas de todas las relaciones, se lleva a cabo una concatenación de las mismas, devolviendo una lista de todas las parejas de soluciones del tipo *(puntuación tripleta correcta, puntuación tripleta incorrecta)*.

Al tener el modelo esta forma, donde los parámetros de todas las relaciones se concatenan a lo largo de una dimensión adicional formando una única matriz para cada uno de los parámetros, se permite que el modelo explote las correlaciones existentes entre las distintas relaciones. Además, permite que el modelo sea mucho más escalable, al tratarse de una concatenación de sub-modelos, de manera que si se deseara introducir una nueva relación al modelo, puede aprovecharse el modelo entrenado previamente e incorporarle la nueva relación, sin necesidad de entrenarlo nuevamente desde cero.

Ya que el modelo no se comporta igual durante el entrenamiento que durante la evaluación, es necesario hacer una función *forward* dual, de manera que mediante un parámetro se ha de especificar si la iteración que se va a realizar corresponde, o no, al entrenamiento. Mientras que en el entrenamiento, el *batch* se compone de tripletas correctas y corruptas, las cuales son operadas por separado, obteniendo así las puntuaciones de las tripletas correctas por una parte y las de las tripletas corruptas por otra, en la evaluación no se hace esta distinción, de manera que el valor de salida del modelo son las puntuaciones de todas las tripletas recibidas y su etiqueta real (correcta o incorrecta).

La implementación desarrollada para la red de neuronas tensorial en Pytorch puede encontrarse en el anexo A.

6.3.3. Función de pérdida

Tras codificar y diseñar el modelo en Pytorch, es necesario implementar la función de pérdida especificada [86], que permite calcular el error producido por el modelo y, consecuentemente, actualizar los pesos del modelo mediante el algoritmo de retropropagación.

De acuerdo con lo expuesto en la sección 4.3.1, la función de pérdida a implementar es la representada en la función 3. Ya que esta función no está incluida por defecto, será necesario implementarla desde cero. Siguiendo las reglas de diseño de Pytorch, la función de pérdida debe ser implementada como una instancia de la clase *Module*, es decir, de la misma manera que el modelo. Por tanto, es necesario especificar tanto la forma de inicialización como el conjunto de operaciones que realiza el modelo dada una entrada para obtener la salida (función *forward*).

La función recibe como parámetros las predicciones obtenidas por el modelo, el valor del parámetro de regularización establecido y el conjunto de parámetros de la red. Estos valores se procesan de acuerdo con la función 3, obteniendo un único valor de salida que es minimizado progresivamente durante el proceso de entrenamiento.

Pytorch incluye de manera implícita el algoritmo de retropropagación, por lo que tras calcular el error, la retropropagación del gradiente del error se realiza de manera automática sobre todos los parámetros del modelo.

La implementación desarrollada para la función de pérdida en Pytorch puede ser consultada en el anexo B.

6.3.4. Método de evaluación

Finalmente, falta determinar el método de evaluación del modelo. De acuerdo con el trabajo de Socher [86]: *dada una tripleta* (e_1, R, e_2)*, se considerará que es cierta si la puntuación devuelta por el modelo para la misma es mayor que el valor límite* T_R *asociado a dicha relación.*

Por tanto, es necesario obtener los valores límite, o *thresholds* de cada una de las relaciones para poder evaluar el modelo. En el algoritmo 1 se muestra el proceso de obtención del threshold de una relación, el cual es repetido tantas veces como relaciones haya.

Algorithm 1 Obtención de thresholds

Parámetros: ($results$ = conjunto de puntuaciones obtenidas por el modelo y etiquetas reales de los datos)

$best\_accuracy = -1$
$best\_threshold = 0{,}0$
$interval = 0{,}001$
$scores = results[0]$
$labels = results[1]$
$min\_score =$ **min** $(scores)$
$max\_score =$**max** $(scores)$
$score\_temp = score\_min$

while $(score\_temp <= score\_max)$
  $predictions = (scores >= score\_temp) * 2 - 1$
  $temp\_accuracy =$**mean**$(predictions == labels)$

  if $temp\_accuracy > best\_accuracy$ **then**
    $best\_threshold = score\_temp$
    $best\_accuracy = temp\_accuracy$

  $score\_temp = score\_temp + interval$
return $best\_accuracy, best\_threshold$

De acuerdo con los autores, para obtener los valores límite de las relaciones, ha de emplearse el conjunto de validación. Este conjunto, al igual que el conjunto de evaluación, se compone de una serie de tripletas (e_1, R, e_2) etiquetadas con un valor que determina si dicha tripleta es, o no, correcta. El número de tripletas correctas e incorrectas está completamente balanceado, es decir, por cada ejemplo

Parámetro	**Valor**
Número de iteraciones	500
Tamaño de *batch*	20.000
Número de tripletas corruptas por tripleta	10
Dimensión de los vectores de embedding (d)	100
Número de rodajas del tensor (k)	3
Valor del parámetro de regularización	1e-4
Entrenamiento de *embeddings*	True

Tab. 4: Parámetros empleados por Socher et al. [86] para el entrenamiento del modelo.

correcto (e_1, R, e_2), se ha generado un ejemplo (e_1, R, e_c) etiquetado como incorrecto, obteniendo por tanto el mismo número de ejemplos correctos que incorrectos.

El objetivo de este algoritmo es maximizar el porcentaje de acierto, y presenta una convergencia estable, independientemente del número de datos. Se han probado otro tipo de aproximaciones, como emplear como valor límite el *equal error rate*. Este valor resulta intuitivamente más adecuado, ya que determina el punto donde se equilibran el porcentaje de elementos incorrectos clasificados como correctos, con el porcentaje de elementos correctos clasificados como incorrectos, por lo que es un valor altamente discriminante. Sin embargo, el cálculo de este valor es muy sensible a la cantidad y al rango de los datos, por lo que si el conjunto de validación ofrecido para una relación es pequeño, este valor estará excesivamente ajustado a este conjunto, por lo que no podrá ser usado en el proceso de evaluación.

6.4. Fase 2: Replicación de los resultados originales

En esta primera fase, el objetivo es llevar a cabo una replicación de los experimentos propuestos por Socher et al [86], para determinar la veracidad de los resultados propuestos por los autores. De la misma manera que el código, y los conjuntos de datos, está también disponible el fichero con los parámetros de entrenamiento empleados por los autores para generar los resultados reflejados en el artículo, así como los valores de los hiperparámetros especificados en la sección 6.3.1. Los valores de todos los parámetros empleados en ambos experimentos se recogen en la tabla 4.

Adicionalmente, en el trabajo se especifica que el algoritmo de optimización empleado es L-BFGS (*Limited Broyden-Fletcher-Goldfarb-Shanno Algorithm*), al cual se le limita el número máximo de optimizaciones por *batch* a 5.

La única diferencia existente en todo el proceso de replicación en nuestros experimentos frente a los propuestos es que, en vez de emplear vectores de palabra, se emplean vectores de entidad durante todo el proceso, y son éstos los vectores que se entrenan de manera paralela al modelo. En el trabajo original, la matriz E no se compone de los vectores asociados directamente a las entidades, sino que corresponden a las palabras del vocabulario que compone, a su vez, las entidades. De esta manera, el vector de *embedding* de una entidad se obtiene mediante la media

aritmética los vectores de las palabras que la componen. En el ejemplo de la figura 12, el vector de la entidad *bengal_tiger*, se compone por la media aritmética entre los vectores de las palabras *bengal* y *tiger*. En cada iteración se actualizan los vectores de palabras y, ya que las operaciones del modelo se llevan a cabo sobre entidades, es necesario componer los vectores correspondientes entidades a partir de sus palabras antes de operar. Esta operación de recomposición en cada iteración acarrea un gran coste computacional, debido a la cantidad de memoria que precisa, además de ralentizar notablemente el proceso. Debido a las limitaciones existentes tanto en capacidad de cómputo como en tiempo, se decide utilizar vectores de entidad.

Ya que la matriz de *embeddings* inicial ofrecida por los autores se trata de una matriz de palabras, previo a entrenar el modelo se lleva a cabo la composición de todas las entidades en la manera explicada anteriormente. Esto da lugar a una matriz E en la que por cada entidad existe una representación vectorial, de manera que durante el entrenamiento esta representación varía, pero siempre está asociada a la misma entidad, y no es necesario llevar a cabo recomposiciones de manera recurrente. El emplear esta aproximación, tal y como indican los autores, supone un sacrificio en la capacidad predictiva del modelo, por lo que se espera que los resultados obtenidos tras la simulación sean inferiores a los expuestos en el trabajo original.

6.4.1. Replicación resultados en WordNet

En primer lugar, se lleva a cabo una replicación del experimento con los datos ofrecidos por los autores para WordNet. Tras entrenar el modelo con los parámetros indicados anteriormente, y obtener los valores de los *thresholds* asociados a cada relación empleando el conjunto de validación, se lleva a cabo una evaluación del modelo con el conjunto de test, obteniendo los resultados recogidos en la tabla 5 para cada una de las relaciones que componen la base de conocimiento. Adicionalmente, se recogen algunas métricas habituales sobre los resultados del total de relaciones en la tabla 6.

Estos resultados son notablemente inferiores a los mostrados en el trabajo original, si bien en él se especifica que el porcentaje de acierto total obtenido por los autores empleando vectores de entidad es cercano a un 70 %, el cual es un 15 % menor al reportado empleando vectores de palabras.

6.4.2. Replicación resultados en Freebase

De la misma manera que en *WordNet*, se repite el mismo proceso de experimentación, esta vez para los conjuntos de datos de *Freebase*. En este caso, se presenta una particularidad en los propios conjuntos de datos, ya que, a pesar de que en el conjunto de entrenamiento aparecen las trece relaciones a predecir, se observa tanto en los conjuntos de validación como de evaluación, aparecen hechos de únicamente siete de las relaciones totales.

De acuerdo con los autores, hay ciertas relaciones que, a nivel computacional, no tiene sentido intentar predecir, ya que a nivel humano tampoco es posible hacerlo. Relaciones como por ejemplo *place_of_birth* o *place_of_death*, las cuáles son imposibles

Relación	**% acierto**
has_instance	0.502
type_of	0.49
member_meronym	0.49
member_holonym	0.54
part_of	0.49
has_part	0.5
subordinate_instance_of	0.51
domain_region	0.5
sysnet_domain_topic	0.47
similar_to	0.42
domain_topic	0.42
TOTAL	0.5

Tab. 5: Resultados obtenidos por el modelo para el conjunto completo de relaciones de *WordNet*

Métrica	**Valor**
Accuracy	0.5
Precission	0.5
Recall	0.81
ROC Area	0.5

		Predicted Value	
		-1	**1**
True Value	**-1**	487	2122
	1	398	2211

Tab. 6: Métricas obtenidas por el modelo para el conjunto de datos de WordNet

Relación	**% acierto**
gender	0.86
nationality	0.32
profession	0.52
place_of_death	N/A
place_of_birth	N/A
location	N/A
institution	0.55
cause_of_death	0.5
religion	0.65
parents	N/A
children	N/A
ethnicity	0.43
spouse	N/A
TOTAL	**0.61**

Tab. 7: Resultados obtenidos por el modelo para el conjunto de relaciones de *Freebase*. Las relaciones marcadas como N/A son aquellas para las que los autores no han incluido ejemplos de validación ni de evaluación.

Métrica	**Valor**
Accuracy	0.61
Precission	0.58
Recall	0.75
ROC Area	0.61

		Predicted Value	
		-1	**1**
True Value	**-1**	10284	12209
	1	8783	16190

Tab. 8: Métricas obtenidas por el modelo para el conjunto de datos de Freebase

de predecir a nivel humano, tampoco son posibles de predecir a partir del resto de hechos de la base de conocimiento y, en cualquier caso, la predicción obtenida podría ser cierta a nivel computacional, pero no en la realidad. Por tanto, estas relaciones aparecen únicamente durante el entrenamiento, de manera que se explote la información existente en las correlaciones ya que permite caracterizar mejor a la entidad, pero no aparecen en los conjuntos de validación ni de evaluación. Las métricas de acierto por relación aparecen recogidas en la tabla 7, mientras que en la tabla 8 se recogen algunas métricas adicionales sobre el modelo completo.

Estos resultados, si bien son mejores que los obtenidos en *WordNet*, siguen estando alejados de los mostrados en el trabajo original, donde el porcentaje de acierto obtenido empleando vectores de entidad tiene un valor cerca del 85 % de acierto.

También se observa que el modelo tiende a predecir la mayoría de las tripletas como incorrectas, lo que puede estar provocado por el empleo de un número demasiado elevado de tripletas corruptas por tripleta durante el entrenamiento. Al generar un número $N\_corruptas$ de parejas del tipo $(tripleta\_correcta, tripeta\_corrupta_N)$, en la que sólo se varía la tripleta corrupta y no la correcta, aparece mucha más variedad

y muchos más ejemplos de tripletas incorrectas que correctas durante el entrenamiento, lo que hace que, consecuentemente, éstas presenten mucha más variedad en sus puntuaciones recibidas por la red. Ya que estas puntuaciones no responden a un patrón concreto, resulta complicado determinar cuál es el *threshold* que delimita la separación entre las puntuaciones correspondientes a las tripletas correctas, y las correspondientes a las tripletas erróneas, lo que afecta directamente a la capacidad predictiva del modelo.

6.4.3. Discusión de resultados

Tras analizar y comparar los resultados obtenidos con los originales, se proponen varios motivos que permitan explicar las diferencias existentes. En primer lugar, se observa que existen ciertas incoherencias entre los parámetros reportados en el artículo y los presentados en el código original. Particularmente, hay una diferencia en el número de rodajas del tensor, el cual tiene un valor de 3 en el fichero de parámetros presentado para el código, mientras que en artículo se dice que el número de rodajas empleado es de 4. Esto puede suponer una variación notable en los resultados, ya que afecta directamente a la arquitectura del modelo predictivo. Además, aumentar el número de rodajas del tensor está ligado de manera directa con aumentar la capacidad de abstracción del mismo, lo que permite extraer más información de las correlaciones y, consecuentemente, realizar mejores predicciones. Por tanto, esta puede ser una causa potencial de introducción de error.

Otro aspecto que puede haber influido negativamente en la reproducción del experimento es la elección del tamaño del *batch*. Como se expone en la sección 2, el tamaño del *batch* está ligado directamente con el correcto aprendizaje, ya que un *batch* demasiado grande hace que el aprendizaje del modelo no sea constante, sino que presente oscilaciones, y que, generalmente, no permita converger a la solución óptima. En el artículo, no se da el valor usado para este parámetro, pero sí se dice que se ha usado *mini-batch*, por lo que se espera que este valor no sea muy elevado. Sin embargo, en el fichero de parámetros, se refleja que este valor es 20.000, lo que implica que, en el caso de *WordNet*, al menos un quinto del total de los datos de entrenamiento existentes está siendo procesado en cada iteración. Esto afecta también negativamente al algoritmo de optimización empleado, L-BFGS, en el cual dentro de cada iteración de entrenamiento, se realizan N sub-iteraciones sobre el propio *batch*, reduciendo su pérdida progresivamente, es decir, se realiza una optimización local para un conjunto de datos. Al emplear un tamaño tan grande, en cada iteración el modelo sobreajusta al subconjunto dado, de manera que al cambiar de batch, el modelo tiene que volver a ajustarse al nuevo conjunto, lo que provoca grandes oscilaciones a lo largo del entrenamiento.

Además de estos motivos, se sabe de manera previa que la precisión del modelo mejora muy notablemente cuando se emplean vectores de palabras, en vez de vectores de entidades. Ya que en este trabajo se ha optado por tomar la segunda aproximación, se esperaba que, consecuentemente, los resultados obtenidos tras la replicación fueran peores que los reportados en el artículo.

Adicionalmente, al disponer de implementaciones externas que llevan a cabo la

replicación de la experimentación en *WordNet*, se puede hacer uso de las mismas para comprobar si es la propia implementación realizada la que introduce el error, o se trata de una combinación de los factores externos expuestos en los párrafos anteriores. Para ello, se lleva a cabo una ejecución del código desarrollado por el usuario Siddhart Agrawal [85], desarrollado en Python, el cual emplea vectores de palabra y los mismos parámetros que los presentados en el trabajo original, por lo que se espera que, en este caso, el porcentaje de acierto final obtenido sea del 86 %, tal y como se reporta en el artículo original. Sin embargo, tras la ejecución, se obtiene un 78 % de acierto, cerca de un 10 % menos de lo reportado originalmente. Esto implica que, además del error introducido al emplear vectores de entidad en vez de vectores de palabra, hay un porcentaje alto de error que viene introducido tanto por los parámetros como por los datos seleccionados en cada iteración, tal y como se hipotetizaba anteriormente.

6.5. Fase 3: Introducción de información ontológica en el modelo predictivo

Conocido el funcionamiento del modelo sin emplear información ontológica sobre las bases de conocimiento a emplear, se proponen una serie de experimentos en los que se incorpore esta información de la manera explicada en la sección 6.2.2. Con esto, se propone comprobar si, efectivamente, incluir este tipo de información ayuda al modelo a mejorar sus predicciones.

Para comprobarlo, se llevan a cabo tres experimentos distintos.

6.5.1. Experimento 1: Predicción de relaciones en Freebase empleando WordNet como ontología

Como primera aproximación a la resolución del problema, se propone un experimento a pequeña escala que permita comprobar previamente si, en efecto, introducir información ontológica mejora los resultados obtenidos por el modelo, o al menos, no hace que estos resultados empeoren.

Para ello, en un intento por optimizar los recursos disponibles, se propone usar *WordNet* como información ontológica, y emplear *Freebase* como base de conocimiento sobre la que predecir. Se determina esto debido a que las relaciones modeladas en *Freebase* son más interpretables a nivel humano, y por tanto más apropiadas para inferir restricciones de tipo sobre las mismas; mientras que como se expone en la sección 6.2.4, *WordNet* tiene un carácter pseudo-ontológico, por lo que resulta más apropiada para ser utilizada como conocimiento ontológico. Además, en ambas bases de conocimiento se conocen los *embeddings* de las entidades de las mismas, lo que evita el tener que buscar una fuente externa de representación y permite partir de unas representaciones que se sabe de manera previa que son válidas.

	Entrenamiento	**Validación**	**Test**
profession	144	34	34
location	45	10	10
cause_of_death	22	4	4
gender	115	28	28
place_of_death	40	8	8
nationality	29	6	6
place_of_birth	34	8	8
religion	14	2	2

Tab. 9: Número de tripletas resultantes por relación tras la división del conjunto total de hechos en los tres conjuntos de entrenamiento, validación y test. En los ejemplos de validación y evaluación se incluyen tanto hechos correctos como incorrectos.

6.5.1.1. Metodología

En primer lugar, necesitamos generar los conjuntos de datos (entrenamiento, validación y test) para el experimento, así como el conjunto de entidades, relaciones y la información ontológica de las entidades. Para ello, ya que vamos a emplear tanto *WordNet* como *Freebase*, necesitamos encontrar aquellas entidades que sean comunes a ambas, de manera que se tenga información tanto de los hechos en los que dicha entidad aparece en *Freebase*, como en *WordNet* para poder extraer su información. Tras realizar esta búsqueda cruzada, se obtiene un total de 1143 entidades.

A continuación, hay que extraer aquellas tripletas del conjunto total de hechos de *Freebase* en el que aparezcan únicamente las entidades de este conjunto. Del total de trece relaciones posibles existentes en *Freebase*, sólo ocho de ellas tienen algún hecho en el que tanto el sujeto como el objeto de la tripleta pertenecen al conjunto reducido de 1143 entidades. Por cada conjunto de hechos de cada relación, llevamos a cabo una partición en los tres conjuntos: entrenamiento, validación y test. Un 90 % de los hechos se emplean para entrenamiento, un 5 % para validación y el 5 % restante para test. Ya que los conjuntos de validación y test se componen de una tripleta incorrecta por tripleta correcta, y nuestro conjunto de hechos sólo tiene tripletas correctas, llevamos a cabo la generación de tripletas corruptas mediante reemplazo aleatorio de la entidad objeto por otra del conjunto de entidades. El número de tripletas resultantes de este proceso por relación se recoge en la tabla 9.

Tal y como se observa, son conjuntos muy pequeños, particularmente en algunas relaciones, por lo que se decide eliminar dichas relaciones al tener un número extremadamente insuficiente de datos. Del total de relaciones, se decide emplear únicamente cinco de ellas: *profession, location, gender, place_of_death* y *place_of_birth*. Aunque en la replicación de resultados originales en *Freebase* elimina las relaciones *place_of_death* y *place_of_birth* en los conjuntos de validación y test, se decide mantenerlos en este caso. Para ello, se parte del enfoque de que no se busca predecir la entidad que actúa como objeto, sino determinar si un hecho dado es cierto, o no.

En cuanto a los parámetros de entrenamiento empleados, al reducir en gran

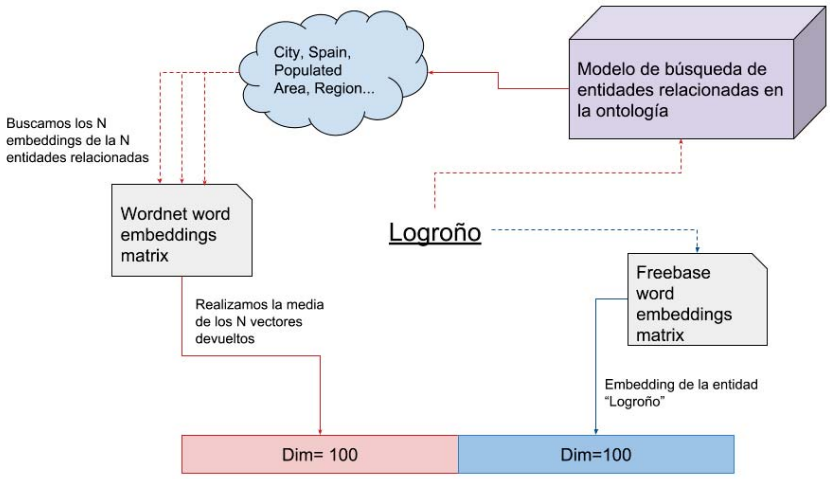


Fig. 18: Proceso de obtención de la información ontológica de las entidades del subconjunto empleado.

medida el conjunto de entrenamiento, se reducen también tanto el tamaño de *batch* como el número de iteraciones, los cuales pasan a ser 400 y 100, respectivamente. Además, se reduce el número de tripletas corruptas por tripleta a 5, para evitar que el modelo sobreajuste a los ejemplos negativos. El parámetro de regularización toma un valor de 1e-2.

6.5.1.2. Generación de la información ontológica

En este experimento se incorpora información ontológica, lo que implica que, antes de iniciar el entrenamiento y para incorporar esta información de la manera indicada en la sección 6.2.2, es necesario tener tanto la matriz con los *embeddings* de las clases, como el conjunto de clases asociado a cada una de las entidades. Tanto la matriz de *embeddings* de las clases, O, como la matriz de *embeddings* de las clases, E, son conocidas previamente, al haber sido generadas durante la replicación de los experimentos de *WordNet* y *Freebase*, respectivamente.

Por tanto, falta determinar, por cada una de las 1143 entidades, cuales son las entidades en *WordNet* con las que se relaciona mediante las relaciones *has_instance* y *type_of*. Estas entidades de *WordNet* serán consideradas las clases. Además, existe también la relación *subordinate_instance_of*, que representa la misma relación que *has_instance* pero en la dirección inversa, por lo que debido a esta redundancia se decide emplear únicamente *has_instance*.

Estas relaciones permiten extraer toda la jerarquía de clases de cada una de las entidades, de la manera ilustrada en la figura 18. Como se ilustra en la figura, una

tripleta en la que aparece la entidad *Logroño* con alguna de las relaciones anteriores es *city, has_instance, logroño*, que nos indicaría que *Logroño* es una instancia de la clase *CIUDAD*, por lo que *CIUDAD* pasaría a formar parte del conjunto de clases que caracterizan a la entidad *Logroño*. A continuación, se buscaría de qué clases es instancia o tipo la clase *CIUDAD* y, en caso de existir, se incorporarían también estas clases a conjunto. A continuación, se repetiría este proceso de búsqueda para las nuevas clases incorporadas. Este proceso de búsqueda se realiza tres veces, ya que se ha observado que a partir de ese punto las clases obtenidas se vuelven excesivamente genéricas.

A continuación, se expone de manera pormenorizada el proceso de obtención de información ontológica para cada una de las entidades.

1. Buscamos las apariciones de la entidad e en el conjunto de hechos de *WordNet* en las tripletas correspondientes a las relaciones *has_instance* y *type_of*. Por cada uno de los hechos extraídos:

    1.1. Si la entidad e actúa como objeto en el hecho de la relación *has_instance*, la entidad c que actúa como sujeto se considera como clase y se incorpora al conjunto de clases C.

    1.2. Si la entidad e actúa como sujeto en el hecho de la relación *type_of*, la entidad c que actúa como objeto se considera como clase y se incorpora al conjunto de clases Cs.

2. Del conjunto de clases obtenido tras las operaciones anteriores, se eliminan todas las duplicidades existentes.

3. Sobre el conjunto sin duplicidades de las clases con las que se relaciona la entidad e, se repite de manera el paso 1. y los pasos 1.1. y 1.2. sobre cada una de las clases C del conjunto obtenido. Este proceso se repite nuevamente sobre el conjunto resultante.

4. Una vez obtenido todo el conjunto de clases C asociado a la entidad e, se obtiene, para cada una de las clases contenidas en C, el vector de*embedding* correspondiente a la misma. Al emplear *WordNet*, el vector de la clase c será el vector de la entidad homónima en *WordNet*.

5. Obtenidos todos los vectores de *embedding* de todas las clases, se realiza la media de los mismos para generar un único vector.

6. El vector generado se almacena en la matriz de clases de cada entidad, en la entidad e correspondiente.

De esta manera, obtenemos una matriz O de las mismas dimensiones que la matriz de *embeddings* E, en la cual por cada una de las entidades, se tiene la media de los vectores de todas las clases que la representan. Las entidades que son similares entre sí reciben vectores de información ontológica similares.

	Sin info. ontológica					Con info. ontológica				
PRUEBA	1	2	3	4	5	1	2	3	4	5
profession	0,5	0,47	0,41	0,53	0,52	0,52	0,55	0,5	0,5	0,5
location	0,5	0,5	0,5	0,5	0,6	0,5	0,3	0,5	0,5	0,5
gender	0,57	0,5	0,5	0,61	0,64	0,64	0,5	0,43	0,4	0,61
place_of_death	0,5	0,38	0,38	0,5	0,5	0,5	0,38	0,63	0,5	0,5
place_of_birth	0,5	0,5	0,25	0,5	0,75	0,25	0,25	0,5	0,5	0,63

Tab. 10: Resultados obtenidos en las cinco pruebas realizadas.

6.5.1.3. Resultados

Al ser un conjunto pequeño, el tiempo que se tarda en entrenar es muy bajo, lo que permite realizar múltiples pruebas, en este caso un total de cinco. Cada prueba se compone de dos etapas, una primera etapa donde se entrena el modelo sin incorporar información ontológica, y una segunda en el que se incorpora esta información. Tras realizar todas las pruebas, se obtienen los resultados reflejados en la tabla 10.

Como se observa, los resultados varían notablemente de una prueba a otra, debido a que el modelo sobreajusta a los datos de entrenamiento, por lo que es muy sensible tanto a la calidad de las tripletas corruptas generadas durante el entrenamiento, como a las tripletas seleccionadas aleatoriamente en cada iteración.

Además, dado que tres de las relaciones se dan entre entidades que representan personas y entidades que representan lugares, es muy probable que, incluso al llevar a cabo reemplazo aleatorio en el entrenamiento, debido a la gran cantidad de entidades del conjunto que pueden ser válidas, la tripleta corrupta generada sea, en efecto, correcta, lo que va en detrimento del correcto aprendizaje del modelo.

En cualquier caso, se observa que los resultados en ambos casos (empleando y sin emplear información ontológica) son muy similares, lo que implica que, si bien no confirma que introducir esta información mejora los resultados, se sabe que al menos no afecta negativamente a la capacidad predictiva del modelo.

6.5.2. Experimento 2: Empleo de las relaciones pseudo-ontológicas de WordNet como información ontológica y predicción sobre el resto de relaciones

Ya que el experimento anterior no permite arrojar resultados concluyentes, debido a que el conjunto de datos de entrenamiento es demasiado pequeño como para entrenar un modelo suficientemente robusto, se propone un nuevo experimento en el que se emplea únicamente *WordNet*.

Además de aportar nuevos resultados que nos permitan seguir estudiando las mejoras inferidas de introducir información ontológica explícita, este experimento también permite estudiar el grado de influencia de las relaciones *has_instance, type_of* y *subordinate_instance_of* en los resultados del modelo original. Ya que en el trabajo original no se realiza esta distinción entre información ontológica e in-

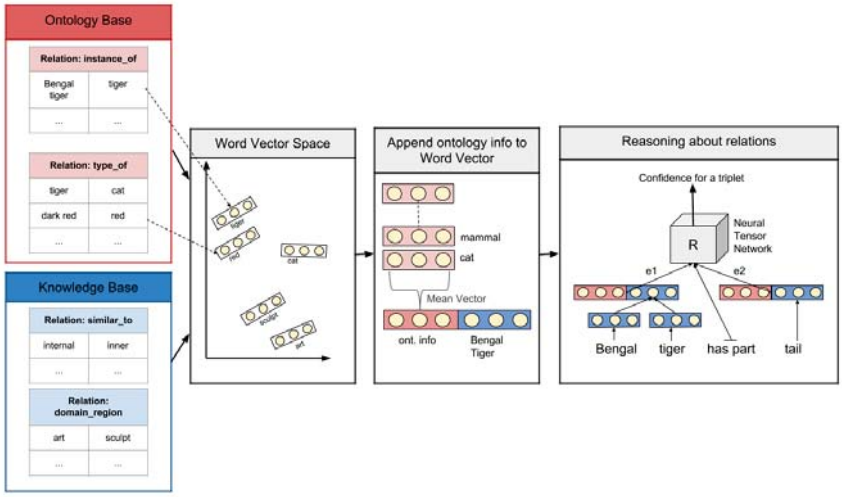


Fig. 19: Diagrama explicativo del experimento realizado sobre *WordNet*. En rojo, se representan las relaciones empleadas para obtener la información ontológica, y en azul, las relaciones a predecir.

formación extraída de correlaciones, resulta interesante comprobar si realmente las correlaciones existentes entre los hechos de estas tres relaciones influyen, o no, en la capacidad predectiva del modelo.

6.5.2.1. Metodología

Como se menciona en la sección 6.2.2, dentro de *WordNet* existen tres relaciones que representan conocimiento de una manera muy similar a una representación ontológica, mientras que el resto son completamente distintas. Por tanto, de acuerdo a este criterio, podríamos llevar una agrupación de las relaciones, separando por un lado las que representan información ontológica, y por otro lado el resto de relaciones, sobre las que se desea predecir.

De la misma manera que se realiza una distinción entre las relaciones que se emplean para generar la información ontológica y las que se van a usar para predecir, es necesario extender esta distinción a las entidades, dividiéndolas así entre entidades que van a actuar como *clases* y entidades sobre las que se va a predecir. De esta manera, se evita que una entidad a predecir pueda aparecer como clase de otra entidad a predecir, lo que mantiene la coherencia en las predicciones. El número de tripletas resultantes por relación tras realizar esta operación se recoge en la tabla 11.

Generada la lista de entidades sobre la que se va a predecir, y conociendo de manera previa la matriz E de *embeddings* para todas las entidades de *WordNet*, se realiza para cada entidad e a predecir el proceso de extracción de información ontológica definido en 6.5.1.2, obteniendo finalmente una matriz de clases O en la cual, por cada entidad e, se tiene un vector que representa la media de las clases que la generan. En la figura 19 se representa de manera conceptual el proceso seguido de división de relaciones, generación de información ontológica y, finalmente, predicción.

	Entrenamiento	**Validación**	**Test**
member_meronym	4941	472	2061
member_holonym	4944	575	2205
part_of	4482	317	1198
has_part	4077	268	1256
domain_region	3240	151	555
sysnet_domain_topic	3068	135	601
domain_topic	3439	156	713
TOTAL	**25123**	**1940**	**7988**

Tab. 11: Especificación del número de tripletas por relación empleadas en el experimento 6.5.2.

Relation	**Sin Info. Ontológica**	**Con Info. Ontológica**
member_meronym	0.51	0.53
member_holonym	0.52	0.5
part_of	0.52	0.54
has_part	0.52	0.5
domain_region	0.49	0.56
sysnet_domain_topic	0.44	0.53
domain_topic	0.51	0.57
TOTAL	**0.51**	**0.53**

Tab. 12: Resultados obtenidos por ambos modelos (con y sin información ontológica) para los conjuntos de datos de test de *Wordnet* para cada una de las siete relaciones a predecir.

6.5.2.2. Resultados

En este caso, ya que el conjunto de datos tiene una dimensión mayor, se decide emplear los parámetros del experimento original, recogidos en la tabla 3. Al igual que en el caso anterior, se entrena en primer un lugar sobre el conjunto de datos de entrenamiento sin incluir información ontológica en el modelo, obligando al modelo a aprender únicamente de las correlaciones existentes entre los hechos. A continuación, se entrena otro modelo con los mismos conjuntos de datos, pero concatenando a cada una de las entidades de cada uno de los hechos su vector de información ontológica correspondiente.

En la tabla 12 se recogen los resultados obtenidos por ambos modelos. Al ser una base de conocimiento léxico, resulta complejo estudiar hasta qué punto pueden mejorar los resultados mediante la introducción ontológica, al no existir una restricción abstracta de tipos en las relaciones como las que se desea inferir. Por ejemplo, no está establecida de manera explícita la restricción de qué tipos de entidades actúan como sujeto y cuáles actúan como objeto en la relación *part_of*, ya que tanto los hechos $(hand, has\_part, finger)$ y $(country, has\_part, city)$ son ciertos, y sin embargo, las entidades que en ellos aparecen son completamente distintas. Es decir, esta

relación no se da de manera estricta entre tipos de entidades, sino que se da entre tipos de palabras, de manera que los sujetos representan hiperónimos, mientras que los objetos representan hipónimos.

Los tipos de palabras, así como sus características, no forman parte de la información ontológica representada. Sin embargo, esa información es la que realmente establece las restricciones en las relaciones, por lo que, aunque la información ontológica proporcionada sí que introduce cierta mejora, resulta esperable que dicha mejora no sea especialmente significativa.

En primer lugar, y comparando los resultados obtenidos por el modelo entrenado con el conjunto de relaciones completas con el modelo entrenado sólo con siete de ellas, se observa que los resultados son muy similares, aunque se aprecian mejoras en los resultados obtenidos en algunas de las relaciones, debido posiblemente a que la cantidad de datos de evaluación es meno en el segundo caso es menor.

Por otra parte, se observa también que, de manera general, los resultados mejoran al incorporar información ontológica, ofreciendo mejores resultados incluso en algunos casos de los obtenidos empleando el conjunto concreto, en el que, a nivel teórico, se cuenta con una cantidad de información similar para caracterizar a las entidades.

Estudiando las relaciones cuyas predicciones se ven mejoradas empleando información ontológica, se observa que las relaciones *member_meronym, part_of*, las predicciones mejoran ligeramente empleando información ontológica. Sin embargo, resulta curioso que en las relaciones contrarias a las anteriores, *member_holonym* y *has_part*, obtengan peores resultados al incluir información ontológica, considerando que la relación que representan es análoga a las anteriores, en las que sí se observa mejora.

Cabe destacar, además, las mejoras considerables que se producen en las relaciones *domain_region, domain_topic* y *sysnet_domain_topic*. En este tipo de relaciones, en las que se agrupan palabras que a nivel conceptual son similares entre sí, resulta más relevante la información ontológica introducida, ya que entidades que pertenezcan al mismo dominio pertenecerán a clases similares.

6.5.3. Experimento 3: Predicción sobre relaciones en Freebase empleando la ontología de DBpedia

Aunque el experimento anterior genera resultados que favorecen a la confirmación de la hipótesis, la interpretabilidad de los mismos no resulta trivial, debido a que, por las características de la propia base de conocimiento, no se puede definir de manera explícita una restricción de tipos en las relaciones. Por ello, se propone experimentar sobre el conjunto de relaciones de *Freebase*, debido a que en estas relaciones existe una restricción conceptual de tipos, por lo que esta base de conocimiento resulta más adecuada.

En el experimento 6.5.1, se llevan a cabo predicciones sobre *Freebase* empleando *WordNet* como fuente de información ontológica. Sin embargo, los conjuntos de datos generados de esta forma son demasiado pequeños para entrenar un modelo que

ofrezca unos resultados lo suficientemente representativos como para ser considerados.

Esto plantea, por tanto, un problema: determinar cómo se va a generar el conocimiento ontológico para las entidades de *Freebase*. Para ello, se decide emplear la ontología proporcionada por DBpedia. Como se expone en la sección 4.1, DBpedia se compone tanto de una base de conocimiento como de una ontología, de manera que cada entidad, o recurso, existente en DBpedia, lleva asociado un árbol de clases en su ontología, la cual es pública y puede ser accedida [1]. Otro de los motivos principales de escoger DBpedia frente a otras bases de conocimiento con ontologías, como Wikidata [95], es que las clases de la ontología están expresadas en lenguaje natural, mientras que en el caso de Wikidata, las clases están representadas por un código, lo que resulta mucho menos intuitivo. Además, la ontología ofrecida por DBPedia es significativamente más sencilla que la existente en Wikidata. Finalmente, otra ventaja importante que presenta DBpedia frente a otras opciones, es que consta de una API sobre la cual se pueden realizar consultas de manera directa, sin necesidad de descargar los datos y alojarlos en un servidor para ello. Si bien realizar las consultas de esta manera supone un aumento en el tiempo de respuesta, dados los recursos computacionales disponibles resulta preferible esta opción.

Al emplear una fuente externa para obtener las clases, no se conocen de manera previa los vectores de representación de las mismas, por lo que es necesario una nueva forma de conseguir los vectores de *embedding* que sirvan de representación de las clases. De los métodos de *embedding* estudiados en la sección 3, se decide emplear Word2Vec, al estar integrado en Python y, particularmente, porque se disponen de modelos de preentrenados para codificación de las entidades de *Freebase*. Estos modelos preentrenados de Word2Vec contienen representaciones para más de un millón y medio de palabras, lo que permite obtener representaciones para todas las clases existentes en DBpedia.

6.5.3.1. Preprocesado de los datos

Este experimento requiere de un proceso exhaustivo de extracción y tratamiento de los datos previo al entrenamiento, debido a que en este caso no se tienen de manera previa los ficheros necesarios para extraer el conjunto de clases asociado a cada entidad, ni las representaciones vectoriales de las clases. Este proceso se representa gráficamente en la figura 20, y puede ser dividido en seis tareas, las cuales se detallan a continuación.

Tarea 1: Obtención de los vectores de embedding de los modelos preentrenados de *Word2Vec* seleccionados

Existen dos modelos preentrenados para *Freebase* en *Word2Vec* que se emplean en este trabajo. Un primer modelo que emplea palabras en lenguaje natural como clave para recuperar los vectores, y otro que emplea el código de la entidad.

Por defecto, se intenta obtener el vector de *embedding* asociado a cada entidad empleando el nombre de la entidad como clave en el modelo que emplea lenguaje natural. Si no se encuentra el vector por su nombre, se lanza una consulta a la API

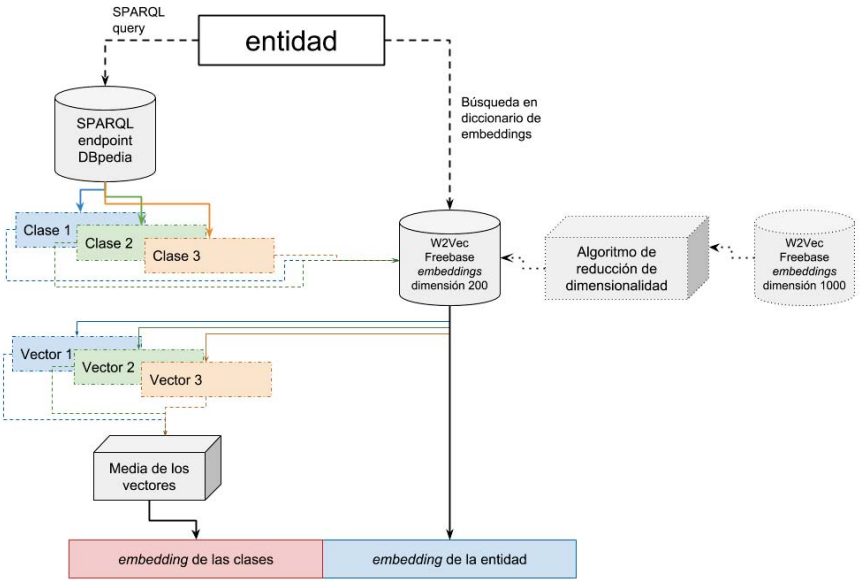


Fig. 20: Esquema general del proceso seguido para generar los vectores de las entidades de *Freebase* con información ontológica extraída de DBpedia.

del *Google Knowledge Graph*, que devuelve el código asociado a dicha entidad. Con este código, se intenta obtener nuevamente el vector de *embedding* realizando una búsqueda con el mismo sobre el modelo que emplea representaciones simbólicas.

Un problema que plantean los modelos de *Word2Vec* seleccionados es que contienen representaciones de dimensión 1000, lo cual es un tamaño excesivo para la capacidad de cómputo disponible. Por ello, se lleva a cabo un procesamiento de ambos modelos por el algoritmo de reducción de dimensionalidad propuesto en [77], lo que permite reducir la dimensión de todos los vectores de ambos modelos de dimensión 1000 a 200. La aplicación de este algoritmo supone una ligera pérdida de información, pero permite que el modelo sea capaz de funcionar correctamente y de manejar dichos vectores.

Tarea 2: Codificación de las clases de la ontología

En primer lugar, tenemos que obtener la codificación vectorial de las clases de la ontología, de manera que luego, cuando obtengamos la lista de clases asociadas a cada entidad, podamos recuperar dichas representaciones y realizar la media entre ellas, obteniendo la codificación de información ontológica asociada a la entidad de la misma manera que se ha realizado en los experimentos anteriores.

Para ello, es necesario en primer lugar obtener los nombres de todas las clases que componen la ontología. Esta lista se obtiene realizando la consulta 1 sobre la API de DBpedia.

```
SELECT DISTINCT *
WHERE {
        ?x a owl:Class .
```

```
}
```

Listing 1: Consulta SPARQL para devolver el conjunto total de clases de la ontología

A continuación, para cada una de las clases, buscamos su codificación vectorial correspondiente en los modelos de dimensionalidad reducida obtenidos tras la tarea anterior. Para aquellas clases cuyo nombre esté compuesto de dos o más palabras, su representación es el resultado de realizar la media aritmética de todos los vectores de cada una de las palabras que aparecen en el nombre de la entidad. Por ejemplo, para la clase *Educational Institution*, su representación será el resultado de realizar la media entre los vectores de las palabras *Educational* e *Institution*. Cada una de las tuplas $(clase, vector)$ es almacenada en un diccionario Python para ser utilizada posteriormente.

Tarea 3: Obtención de la jerarquía de clases de las entidades

Una vez obtenidas las representaciones de las clases, lo siguiente es obtener, para cada una de las entidades existentes en *Freebase*, cuáles son las clases, de acuerdo a la ontología de DBpedia, que la representan.

Para ello, por cada una de las entidades e del conjunto total de entidades de *Freebase*, un total de 74.043, se realiza la consulta 2 sobre la API de DBpedia.

```
PREFIX dbr: <http://dbpedia.org/resource/>

SELECT DISTINCT ?class
WHERE {
        dbr: {entity} rdf:type ?class .
        ?class a owl:Class .
}
```

Listing 2: Consulta SPARQL para obtener la clase de una entidad

Esta consulta nos devuelve la clase a la que pertenece la entidad (o recurso en DBpedia). Sin embargo, ya que queremos no sólo la clase de la que es instancia directamente la entidad, sino toda la jerarquía de clases de la misma, es necesario realizar un proceso recursivo de búsqueda ascendente que nos devuelva todo el conjunto de clases. Para ello, para la clase c obtenida tras realizar la consulta anterior, se realiza una nueva búsqueda mediante la consulta 3, la cual nos devuelve la clase $c\_superior$ de la cual la clase c es una subclase. Este proceso se realiza recursivamente sobre cada una de las clases $c\_superior$ obtenidas en cada iteración, hasta alcanzar la raíz del árbol de clases, es decir, hasta que $c\_superior == owl:Thing$.

```
PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT DISTINCT ?class
WHERE {
        dbr: {clase} rdfs:subClassOf ?class .
        ?class a owl:Class .
```

```
}
```

Listing 3: Consulta SPARQL para obtener la superclase de una clase

De igual manera que en la tarea anterior, por cada una de las entidades se genera una tupla $(entidad, lista\_clases)$. Todas estas tuplas son almacenadas en un diccionario Python para su posterior utilización. Aquellas entidades que no representan un recurso en DBpedia, y que por tanto no es posible obtener su conjunto de clases son registradas en un fichero.

Tarea 4: Obtención de los vectores de *embedding* de las entidades

Ya que se ha decidido emplear representaciones vectoriales de dimensión 200 procedentes de un modelo preentrenado de *Word2Vec* para las clases, se decide también emplear este método para representar las entidades, a fin de mantener la proporción en los vectores resultantes entre la cantidad de información que identifica a la identidad y la cantidad de información ontológica concatenada.

Para ello, se realiza un proceso análogo al realizado en la tarea 2, es decir, por cada una de las entidades e del conjunto total, extraemos del modelo preentrenado su vector de dimensión 200 correspondiente. De igual manera, si la entidad está compuesta por dos o más palabras, el vector que represente a la entidad será la media de todos los vectores de las palabras que la componen. También, al igual que en el caso anterior, todas aquellas entidades para las que no se ha podido generar una representación son anotadas en un fichero.

Tarea 5: Revisión de las entidades descartadas

Tras la realización de la tarea tres, se obtiene una lista con aquellas entidades para las que no ha sido posible obtener su jerarquía de clases, bien por no tener un recurso correspondiente en DBpedia o bien porque no se les ha asignado ninguna clase dentro de dicha base de conocimiento. La lista generada tras la realización de la tarea contiene cerca de 5.000 entidades. Dentro de dichas entidades, podemos distinguir, a grandes rasgos, dos grandes grupos. Por un lado, existe un grupo de entidades que se tratan de nombres propios de personas, las cuales se descartan; sin embargo, en el otro grupo encontramos una serie de entidades que representan nombres de religiones.

En este caso, se observa que ciertas religiones sí existen en DBpedia, mientras que otras no están recogidas. Sin embargo, se observa que todas las entidades que representan religiones han sido clasificadas como instancias de la clase *Ethnic Group*, por lo que, ya que todas ellas pertenecen al mismo tipo, se decide extender esta información a todas aquellas entidades que representan religiones y que no han sido encontradas en DBpedia. De esta manera, estas entidades pasan del conjunto de entidades a descartar al conjunto de entidades a emplear.

De igual manera, tras la realización de la tarea 4 se obtiene también una lista con aquellas entidades para las que no se ha podido obtener, de manera directa, un vector de *embedding*. Tras la observación de esta lista, se comprueba que una gran cantidad de entidades descartadas son nombres de profesiones, mientras que

el resto son, en su mayoría, nombres propios de personas, por lo que se descartan de manera inmediata. Las entidades que representan profesiones son recopilados del fichero total de entidades a descartar. Una vez obtenidas todas las entidades que representan profesiones, se obtienen unas representaciones aproximadas de las mismas aprovechando la propiedad ofrecida por las representaciones de *Word2Vec* de permitir analogías entre palabras. De esta manera, si se conoce el *embedding* de las entidades *singer*, *song* y *book*, y se desea conocer la representación de la entidad *writer*, se podría establecer la analogía *V(writer) = V(book) - V(singer) + V(song)* (escritor es a libro lo que cantante es a canción), de la misma manera que en el ejemplo mostrado en la sección 3.3.

Tras llevar a cabo ambos procesos de descarte, se obtiene un conjunto final de 46.154 entidades.

Tarea 6: Generación de la información ontológica de las entidades

Una vez obtenido el conjunto definitivo de entidades, así como la matriz de *embeddings* E con las representaciones de todas ellas, lo siguiente es componer la matriz O, en la que al igual que en los experimentos anteriores se almacena, por cada entidad, el vector que representa la información ontológica de la misma.

Para ello, dada la entidad e, el diccionario de tuplas $(clase, vector)$ generado en la tarea 2, y el diccionario $(entidad, lista\_clases)$ generado en la tarea 3 se realiza el siguiente procedimiento:

1. Extraer la lista de clases asociada a la entidad e.

2. Por cada una de las clases c perteneciente a $lista\_clases$, extraer su vector de *embedding* del diccionario de representación de clases.

3. Una vez extraídos todos los vectores de todas las clases pertenecientes a $lista\_clases$, realizar la media aritmética de los mismos para generar un vector único.

4. Almacenar este vector en la matriz O en la posición correspondiente a la entidad e.

Tras este proceso, se obtiene una matriz O, en la que por cada una de las entidades del conjunto de *Freebase* existe un vector de dimensión 200 que representa su jerarquía de clases en DBpedia.

6.5.3.2. Metodología

Tras el preprocesado de los datos, con la consecuente reducción del conjunto total de entidades de 75.043 a 46.154, es necesario comprobar el número de ejemplos por relación que existen una vez disminuido el número de entidades.

En primer lugar, se observa que una de las relaciones que desaparece es la relación *gender*, ya que, a pesar de ser la relación más sencilla de predecir, las entidades que pueden aparecer como objeto (*male* y *female*) no pertenecen a ninguna clase. Al

Relation	Sin Info. Ontológica	Con Info. Ontológica
nationality	0.37	0.5
profession	0.47	0.6
institution	0.52	0.5
cause_of_death	0.52	0.53
religion	0.39	0.54
ethnicity	0.5	0.53
TOTAL	**0.45**	**0.53**

Tab. 13: Resultados obtenidos por el modelo empleando y sin emplear información ontológica para el conjunto de test de *Freebase* por cada una de las seis relaciones a predecir

no existir una clase *GÉNERO* en DBpedia que identifique a estas entidades, no se puede generar información ontológica sobre las mismas, por lo que son eliminadas del conjunto de entidades. Al ser estas dos las únicas entidades que pueden aparecer como objeto, la relación desaparecer consecuentemente.

Sobre el resto de relaciones, se procede igual que en la replicación de resultados en *Freebase* realizada en la sección 6.4, en la que sólo se consideran las relaciones *cause_of_death, profession, religion, nationality, institution* y *ethnicity* a la hora de evaluar el modelo.

Los parámetros de entrenamiento del modelo son los mismos que los recogidos en la tabla 4, con la diferencia de que en este experimento la dimensión del vector de *embedding* es de 200.

6.5.3.3. Resultados

Al igual que en el experimento anterior, se entrenan dos modelos distintos: en primer lugar un modelo en el que no se incluye la información ontológica sobre los vectores de las entidades, y a continuación, otro modelo en que se concatena a cada entidad su vector de información ontológica correspondiente. Los resultados se recogen en la tabla

Como se observa, en este caso la mejora obtenida por el modelo empleando información ontológica es mucho más prominente que la obtenida en el experimento anterior, rozando cerca del 8 % de mejora total en las predicciones.

Dentro de las relaciones que presentan una mejora más significativa en sus predicciones al incluir la jerarquía de clases de cada una de las entidades, se encuentran las relaciones *nationality* (cerca de un 13 %), *profession* (cerca de un 14 %) y *religion* (cerca de un 15 %). En el resto de relaciones, los resultados obtenidos son muy similares en ambos casos. Estudiando de manera más pormenorizada aquellas relaciones afectadas de una manera más favorable por la introducción de información ontológica, se observa que, para el caso de la relación *nationality*, todas las entidades que aparecen como sujeto contienen las clases *PERSON* y *AGENT* entre su jerarquía, mientras que aquellas que aparecen mientras que en el caso de las entidades que actúan como objeto, aparecen las clases *COUNTRY, PLACE* y *POPULATED*

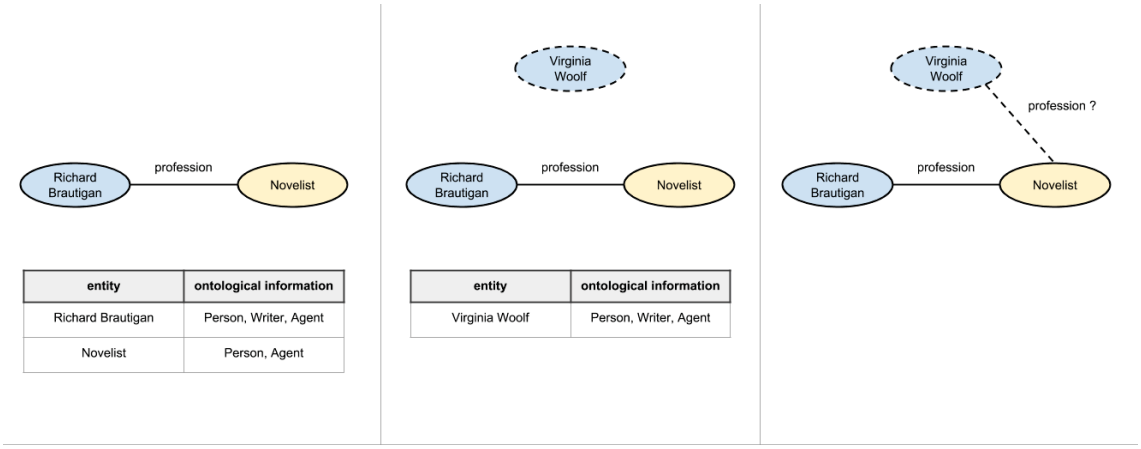


Fig. 21: Diagrama explicativo del proceso de razonamiento del modelo dado un hecho compuesto por una entidad conocida y una entidad nueva.

PLACE. Es decir, hay mucha homogeneidad entre las clases que dan lugar a las entidades sujeto, y mucha homogeneidad entre las clases que dan lugar a las entidades objeto.

De igual manera, esta homogeneidad se da en la relación *profession*, en el que de acuerdo a la ontología de DBpedia, todas las profesiones son instancias de la clase *PERSON*, aunque en algunas esta instancia no se de directamente con la clase anterior, sino con alguna subclase de la misma. Este es el caso de la entidad *actor*, que es instancia de la clase *ARTIST*, la cual es una subclase de la clase *PERSON*. En cualquier caso, en todas las entidades objeto de la relación *profession* se da una lista de clases muy similar, lo que permite al modelo llegar a inferir restricciones acerca de las mismas. Lo mismo sucede en el caso de la relación *RELIGION*, en el que todas las entidades objeto pertenecen a la clase *ETHNIC GROUP*.

Observando aquellas relaciones en las que no se reporta una gran mejora, como la relación *cause_of_death*, se comprueba que, en efecto, existe una gran heterogeneidad entre las clases que dan lugar a las entidades que actúan como objeto. Un ejemplo de entidades objeto válidas de la relación con sus respectivas clases son: *murder* (*MILITARY CONFLICT*), *typhoid_fever* (*DISEASE*) y *firearm* (*DEVICE, WEAPON*). Como se observa, no existen coincidencias entre las clases que representan a las distintas entidades, por lo que esta información no permite inferir una restricción general acerca de los tipos que pueden aparecer en la relación. Esto hace que, de cara a la predicción, la información se extraiga principalmente del *embedding* de la entidad, lo que provoca que los resultados empleando y sin emplear información ontológica sean muy similares en este caso.

6.6. Fase 4: Predicción sobre hechos que incluyen entidades nuevas

Considerando los buenos resultados obtenidos en el experimento anterior, en los que se observa que el modelo ha sido capaz de explotar la información ontológica

proporcionada para inferir restricciones en las relaciones, se propone un experimento para comprobar si las restricciones inferidas son suficientemente buenas como para, dado un hecho compuesto por una entidad conocida y una entidad desconocida, determinar si este hecho es correcto o incorrecto.

Para ello, se parte de la idea de que si el modelo ha sido capaz de explotar la información proporcionada por el conjunto de clases de las entidades para inferir restricciones, puede considerarse que, dada una entidad nueva para la que no se tiene información contextual extraída de las correlaciones pero sí se conoce su información ontológica, el modelo sea capaz de aplicar la restricción asociada a la relación sobre la entidad, permitiendo así razonar sobre la misma. En la figura 21 se expone esta idea de manera ejemplificada.En el ejemplo, se observa que existe una tripleta conocida por el modelo $(RichardBrautigan, profession, Novelist)$, en la que se conoce la jerarquía de clases de las entidades sujeto y objeto. Por otra parte, la entidad $VirginiaWoolf$ es desconocida por el modelo, lo que a nivel teórico no permitiría razonar sobre la misma. Sin embargo, dado que la entidad comparte por completo su jerarquía de clases con la entidad $RichardBrautigan$, se puede establecer la suposición de que, por analogía, el modelo es capaz de establecer razonamientos sobre hechos que incluyan esta nueva entidad. Por ello, debido al parecido entre las entidades $RichardBrautigan$ y $VirginiaWoolf$, existe la posibilidad de que el modelo sea capaz de determinar correctamente la veracidad del hecho $(VirginiaWoolf, profession, Novelist)$.

6.6.1. Metodología

En este experimento, se parte de los mismos conjuntos de datos que en el experimento anterior, de manera que se conocen previamente tanto los vectores de *embedding* de todas las entidades, como el vector que representa la jerarquía de clases de cada una de ellas.

Para generar el conjunto de entidades nuevas, se seleccionan aleatoria 1.500 entidades del conjunto total de entidades conocidas, así como todas las tripletas correctas del conjunto completo de hechos que están formadas por una entidad del conjunto generado de entidades nuevas, y otra del conjunto de entidades conocidas. Ya que los hechos extraídos son todos correctos, se genera mediante reemplazo aleatorio una tripleta incorrecta por cada tripleta correcta. Este conjunto generado será el conjunto que usaremos para evaluar el funcionamiento del modelo sobre entidades nuevas. Los conjuntos de entrenamiento, validación y test se componen únicamente por hechos formados por entidades conocidas.

Además, en este modelo se desactiva el entrenamiento de la matriz de *embeddings* E, de manera que se evite tanto que puedan aparecer duplicidades entre los vectores de *embedding* de las entidades conocidas y de las entidades nuevas, así como para asegurar una mayor igualdad de condiciones entre los dos grupos de entidades. Los *embeddings* de todas las entidades (nuevas y conocidas) están recogidos en la matriz E obtenida tras la tarea 4 del experimento anterior, y de igual manera la matriz de información ontológica empleada, O es la misma que la obtenida tras la tarea 6.

Relation	Solo entidades conocidas	Entidades conocidas y nuevas
nationality	0.53	0.32
profession	0.5	0.5
institution	0.56	0.58
cause_of_death	0.57	0.7
religion	0.55	0.55
ethnicity	0.36	0.32
TOTAL	**0.53**	**0.41**

Tab. 14: Resultados obtenidos por el modelo para el conjunto compuesto únicamente por entidades conocidas y para el conjunto compuesto tanto por entidades conocidas como por entidades nuevas.

6.6.2. Resultados

Se entrena un único modelo empleando información ontológica y sin entrenamiento de la matriz de *embeddings*, con los mismos parámetros que en el experimento anterior, pero reduciendo el número de iteraciones a 300. Esto es debido a que al no ser necesario entrenar la matriz E, que es el parámetro de mayor dimensión, la pérdida del modelo converge de manera mucho más inmediata.

Una vez entrenado el modelo, se prueba sobre los dos conjuntos de test generados: el conjunto de test compuesto por hechos en los que ambas entidades son conocidas y el conjunto compuesto por hechos en los que una entidad es conocida y otra no. Los resultados obtenidos para ambos conjuntos se recogen en la tabla 14.

Observando los resultados, se comprueba que, si bien en algunas relaciones existe una gran diferencia en los resultados obtenidos por las predicciones, como en la relación *nationality*, se comprueba que, en la mayoría de las relaciones, los resultados obtenidos son muy similares en ambos conjuntos. Debido a que no se entrenan las representaciones vectoriales de las entidades, proceso durante el cual el *embedding* de la entidad se va actualizando y ajustando a las correlaciones obtenidas para la misma, la principal fuente de información del modelo en este caso pasa a ser la jerarquía de clases de las entidades. Estos resultados implican también que, de manera aproximada, el modelo ha sido capaz de inferir las restricciones de clases deseadas, de manera que, aunque se presente una nueva entidad sobre la que sólo se conoce su jerarquía de clases, esta información resulta suficiente para establecer un razonamiento sobre la misma.

También cabe destacar que, en algunas relaciones, los resultados obtenidos por el conjunto compuesto por entidades conocidas y nuevas obtiene mejores resultados que el conjunto compuesto enteramente por entidades conocidas. Esto a nivel lógico es inconsistente, ya que debería obtenerse un mayor acierto para el conjunto conocido. Sin embargo, este puede estar provocado por haber realizado un sub-muestreo en los conjuntos de datos para facilitar la comparación. Al realizar la partición de datos de manera que ambos conjuntos de test tuvieran el mismo número de ejemplos por relación, sucede que se reduce en gran medida el número de ejemplos disponibles para entrenamiento y validación, particularmente para éste último. Aunque el conjunto de

validación no es relevante para el aprendizaje, es muy determinante en la evaluación, ya que a partir de él se obtienen los valores límite o *thresholds* para cada una de las relaciones, que permiten determinar si un hecho es, o no, cierto. Por tanto, un conjunto de validación pequeño provoca que el límite obtenido para una relación tenga un carácter muy local, por lo que al aplicar dicho límite obtenido al conjunto de test, los resultados en términos de precisión obtenidos pueden ser muy dispares.

7. CONCLUSIONES Y TRABAJO FUTURO

7.1. Conclusiones

Este trabajo presenta un método de introducción de información ontológica en un modelo de compleción de bases de conocimiento de una manera sencilla y eficiente, además de ofrecer una implementación actualizada de la red de neuronas tensorial empleando la librería Pytorch. Así mismo, se presentan una serie de experimentos en los que no sólo se prueba este método y se demuestran las potenciales mejoras reportadas por la introducción de este tipo de información en el modelo, sino que también se lleva a cabo un análisis de la red de neuronas tensorial, del funcionamiento de la misma y de los factores más determinantes en su correcto funcionamiento.

En cuanto a las contribuciones realizadas, la principal contribución es presentar un método funcional y sencillo de introducción de información ontológica explícita sobre un modelo predictivo. Este método no sólo es aplicable a cualquier ontología, como se refleja en este trabajo, sino que permite la portabilidad entre distintas bases de conocimiento, de manera que se puede seleccionar, dependiendo de la naturaleza de los datos sobre los que se desea razonar, la fuente de información ontológica que mejor se adapte a los mismos. En este trabajo, se han probado dos fuentes de conocimiento ontológico distintas sobre dos bases de conocimiento distintas: *WordNet* y *Freebase*. En el caso de *WordNet*, se ha empleado una partición de la propia base de conocimiento como información ontológica, dadas las características particulares de la misma. Debido a que es una base de conocimiento léxico-semántico, sobre el cual es complicado establecer una restricción de tipos al uso, no se observa una gran mejora en los resultados obtenidos por el modelo incorporando información ontológica. Sin embargo, en algunas relaciones concretas sí se reporta una mejora notable al aumentar la cantidad de información disponible para realizar la predicción, como en la relación *domain_region*, aunque la mejora es, en términos globales, bastante sensible. En el caso de *Freebase*, se observa que, al introducir información ontológica procedente de *DBpedia*, se obtienen resultados muy similares a los obtenidos tras la replicación de resultados del trabajo original. Esto es destacable, considerando que en la replicación de resultados se emplean todas las relaciones en el entrenamiento, lo que permite extraer más información derivada de correlaciones, además de emplear los vectores de *embedding* óptimos, al ser los proporcionados originalmente por los autores, mientras que los empleados en este experimento son obtenidos de un modelo externo de representación del lenguaje (*Word2Vec*).

También, este trabajo, hasta donde se tiene constancia, es uno de los primeros en los que se presenta un método de partición de datos en el que se genera un conjunto de test que contiene entidades nuevas, es decir, que no han sido vistas por el modelo durante el entrenamiento; y en el que se lleva a cabo una evaluación del modelo sobre dicho conjunto de datos. En los trabajos estudiados, no se reportan resultados sobre hechos que incluyan entidades nuevas, debido a que sólo las entidades existentes en la base de conocimiento tienen una representación asociada. En el presente trabajo, se propone la utilización de un método de representación de entidades externo al modelo, de manera que la información asociada una entidad no es extraída

de los hechos de la base de conocimiento en los que aparece, sino que se trata de una información de carácter semántico y contextual. Además, se comprueba que los resultados obtenidos empleando este método de representación son muy similares a los obtenidos empleando las codificaciones generadas *ad hoc* por el modelo predictivo, lo que supone un avance en términos de escalabilidad y portabilidad, al poder emplear el mismo método de representación de entidades para múltiples bases de conocimiento.

En cuanto a contribuciones menores, cabe destacar que se ha realizado una implementación desde cero de la red de neuronas tensorial empleando una de las librerías más novedosas y potentes en el área del aprendizaje profundo en la actualidad: Pytorch. Esta implementación, además, está disponible de manera libre para ser reutilizada por cualquier usuario en el futuro. Finalmente, en este trabajo se ha realizado un estudio detallado del estado de la cuestión sobre las áreas de redes de neuronas artificiales, métodos de representación distribuida del lenguaje y métodos de compleción de bases de conocimiento, el cual ha permitido extraer una serie de conclusiones y limitaciones acerca de las mismas, que se han abordado en este trabajo.

7.2. Trabajo Futuro

En vista de los resultados obtenidos, se plantean una serie de líneas futuras orientadas a la mejora de los mismos. Estas mejoras se pueden enfocar hacia tres áreas distintas.

Una posible línea de mejora se centra en la modificación de la arquitectura de la red de neuronas tensorial y los parámetros empleados. Desde su publicación, en 2013, han aparecido múltiples mejoras dentro del área del aprendizaje profundo, donde se engloba este modelo. Estas mejoras se pueden aplicar sobre la arquitectura de la red, planteando por ejemplo un aumento en el número de capas, ya que actualmente solo consta de una. De acuerdo con las bases del aprendizaje profundo, aumentar el número de capas aumenta la capacidad de abstracción, por lo que es coherente plantear que, quizás aumentando el número de capas se consiga una mejor capacidad de generalización. Conseguir que el modelo sea capaz de generalizar es, además, beneficioso para llevar a cabo razonamiento sobre hechos que incluyan entidades nuevas.

Otro área donde se pueden proponer diferentes mejoras es en la forma de representación tanto de las entidades como de la información ontológica. En este trabajo, esta representación se ha hecho empleando vectores de palabras lo cual, si bien es funcional e intuitivo, resulta poco escalable. Como sucede en el experimento en el que se predice sobre *Freebase* usando la jerarquía de clases de DBpedia, al emplear una codificación basada en palabras puede suceder que alguna de las palabras que forman parte del nombre de la entidad o de la clase no tengan una representación asociada, lo que provoca que la entidad o la clase sea descartada. Una posible solución a este problema es emplear un método de *embedding* que no genere representaciones de palabras, como por ejemplo *FastText*. Esta forma de representación del lenguaje

emplea una codificación por *n-gramas*, es decir, por fragmentos de palabra. Esto hace que sea mucho más escalable, ya que cualquier palabra se puede recomponer a partir del conjunto de *n-gramas* codificado y, consecuentemente, cualquier entidad o clase. Dada la alta portabilidad y escalabilidad de este modelo, así como los buenos resultados que se le reportan, resulta interesante sustituir el método de codificación de palabras empleados por éste, y comprobar la influencia que tiene sobre los resultados.

Otro área donde se proponen mejoras es en la generación de información ontológica. En este trabajo, se han empleado como bases de información ontológica tanto *WordNet* como *DBpedia*. Sin embargo, existen otras muchas bases de conocimiento ontológico, algunas de ellas con una mayor extensión y un mayor nivel de especificidad, como *Wikidata*. Esta base de datos contiene una mayor cantidad de recursos que las empleadas, además de ofrecer una ontología con un mayor nivel de complejidad y, consecuentemente, mas específico. En la ontología de *DBpedia*, se distinguen un máximo de cuatro niveles de profundidad en el árbol de clases, mientras que en *Wikidata* sólo para modelizar las entidades de la clase *GUN*, se distinguen tres niveles de especifidad. Esta ontología ofrece una información mucho más completa acerca de la entidad a representar, pero, por otra parte, emplear un nivel de especificidad tan alto va en detrimento de inferir restricciones de carácter general acerca de las relaciones. Aún con todo, es interesante comprobar qué sucede al pasar de una ontología sencilla a una ontología mucho más compleja.

Por último, también dentro del área de representación de la información ontológica, una mejora que se plantea es llevar a cabo una ponderación de acuerdo con la importancia de las clases en la jerarquía que representa a una entidad. Si se tiene por ejemplo la entidad *Virginia Woolf*, como en el caso anterior, se obtiene que el conjunto de clases que la representan es *AGENT, PERSON, WRITER*. A la hora de codificar la información ontológica de la entidad, todas las clases cobran la misma importancia, aunque, a nivel objetivo, se observa que hay clases que representan mejor a la entidad. En este caso, la clase *WRITER* es la más cercana a la entidad, y la que nos da una información más detallada de la misma, mientras que el conocimiento proporcionado por la clase *PERSON* es mucho más global, y es idéntico para todas las entidades que representan personas, al igual que sucede con la clase *AGENT*. Conseguir una forma de representación de la jerarquía de clases que permita equilibrar el conocimiento específico y el conocimiento general resulta muy interesante, ya que permite caracterizar mejor a la entidad y, potencialmente, mejorar la capacidad predictiva del modelo, particularmente cuando se busque razonar sobre un hecho compuesto por una entidad nueva y una entidad conocida.

Referencias

[1] Dbpedia ontology. `http://mappings.dbpedia.org/server/ontology/classes/`. [Online; accessed July-2018].

[2] Rnn vs feedforward. `https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5`. [Online; accessed July-2018].

[3] Nndb: Tracking the entire world. *Reference Reviews*, 28(1):49–49, 2014.

[4] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski. Connectionist models and their implications: Readings from cognitive science. chapter A Learning Algorithm for Boltzmann Machines, pages 285–307. Ablex Publishing Corp., Norwood, NJ, USA, 1988.

[5] D. D. Alex LeNail, Clare Liu. Reimplementing neural tensor networks for knowledge base completion (kbc) in the tensorflow framework. `https://github.com/dddoss/tensorflow-socher-ntn`, 2015. [Online; accessed July-2018].

[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[7] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural Netw.*, 2(1):53–58, Jan. 1989.

[8] T. Baldwin and M. Lui. Language identification: The long and the short of the matter. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, HLT '10, pages 229–237, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[9] Y. Bengio. *A Connectionist Approach to Speech Recognition*, pages 3–23.

[10] Y. Bengio, H. Schwenk, J.-S. Senécal, F. Morin, and J.-L. Gauvain. *Neural Probabilistic Language Models*, pages 137–186. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[11] J. Bian, B. Gao, and T.-Y. Liu. Knowledge-powered deep learning for word embedding. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 132–148. Springer, 2014.

[12] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. *CoRR*, abs/1607.04606, 2016.

[13] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: A collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, SIGMOD '08, pages 1247–1250, New York, NY, USA, 2008. ACM.

[14] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[15] A. Bordes, J. Weston, R. Collobert, Y. Bengio, et al. Learning structured embeddings of knowledge bases. In *AAAI*, volume 6, page 6, 2011.

[16] H. Bourlard and Y. Kamp. Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4):291–294, Sep 1988.

[17] D. Cai and H. Zhao. Neural word segmentation learning for chinese. *CoRR*, abs/1606.04300, 2016.

[18] D. Chen, R. Socher, C. D. Manning, and A. Y. Ng. Learning new facts from knowledge bases with neural tensor networks and semantic word vectors. *arXiv preprint arXiv:1301.3618*, 2013.

[19] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[20] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The helmholtz machine. *Neural Computation*, 7(5):889–904, 1995.

[21] A. de Gispert, G. Iglesias, and B. Byrne. Fast and accurate preordering for smt using neural networks, 01 2015.

[22] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization, 2010.

[23] C. Fellbaum. Wordnet. an electronical lexical database, 1998.

[24] J. Firth. A synopsis of linguistic theory 1930-1955. *Studies in linguistic analysis*, pages 1–32, 1957.

[25] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Mach. Learn.*, 29(2-3):131–163, Nov. 1997.

[26] A. Gangemi, N. Guarino, C. Masolo, A. Oltramari, and L. Schneider. Sweetening ontologies with dolce. In A. Gómez-Pérez and V. R. Benjamins, editors, *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, pages 166–181, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[27] M. Gardner and T. Mitchell. Efficient and expressive knowledge base completion using subgraph feature extraction. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1488–1498, 2015.

[28] D. Gillick, C. Brunk, O. Vinyals, and A. Subramanya. Multilingual language processing from bytes. *CoRR*, abs/1512.00103, 2015.

[29] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[30] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 315–323, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

[31] Google. Introducing the knowledge graph: things, not strings. `https://googleblog.blogspot.com.es/2012/05/introducing-knowledge-graph-things-not.html`, 2012.

[32] G. Guizzardi and G. Wagner. A unified foundational ontology and some applications of it in business modeling. In *In CAiSE Workshops (3*, pages 129–143, 2004.

[33] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[34] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[35] M. Henderson, R. Al-Rfou, B. Strope, Y. hsuan Sung, L. Lukács, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil. Efficient natural language response suggestion for smart reply. *ArXiv e-prints*, 2017.

[36] G. E. Hinton. Rectified linear units improve restricted boltzmann machines vinod nair.

[37] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7):1527–1554, July 2006.

[38] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[39] G. E. Hinton and R. S. Zemel. Autoencoders, minimum description length and helmholtz free energy. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, NIPS'93, pages 3–10, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[40] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[41] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[42] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359 – 366, 1989.

[43] J.-H. Huang, J.-M. Huang, A.-P. Li, and Y.-Z. Tong. Knowledge reasoning based on neural tensor network. In *ITM Web of Conferences*, volume 12, page 04004. EDP Sciences, 2017.

[44] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov. Fasttext.zip: Compressing text classification models. *CoRR*, abs/1612.03651, 2016.

[45] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016.

[46] N. Kalchbrenner, E. Grefenstette, and P. Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[47] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *Ann. Math. Statist.*, 23(3):462–466, 09 1952.

[48] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[49] T. Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43(1):59–69, Jan 1982.

[50] T. Kohonen and T. Honkela. Kohonen network. *Scholarpedia*, 2(1):1568, 2007. revision #122029.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[52] S. Lai, K. Liu, S. He, and J. Zhao. How to generate a good word embedding. *IEEE Intelligent Systems*, 31(6):5–14, Nov.-Dec. 2016.

[53] T. K. Landauer and S. Dumais. Latent semantic analysis. *Scholarpedia*, 3(11):4356, 2008. revision #91420.

[54] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[55] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, Dec 1989.

[56] J. Y. Lettvin, H. R. Maturana, W. S. McCulloch, and W. H. Pitts. What the frog's eye tells the frog's brain. *Proceedings of the IRE*, 47(11):1940–1951, Nov 1959.

[57] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *AAAI*, volume 15, pages 2181–2187, 2015.

[58] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

[59] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, Dec 1943.

[60] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[61] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *arXiv preprint arXiv:1301.4546*, 2013.

[62] G. A. Miller. Wordnet. a lexical database for english, 1995.

[63] G. A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, Nov. 1995.

[64] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969.

[65] G. Molina, F. AlGhamdi, M. Ghoneim, A. Hawwari, N. Rey-Villamizar, M. Diab, and T. Solorio. Overview for the second shared task on language identification in code-switched data, 01 2016.

[66] C. Morris. The measurement of meaning. charles e. osgood , george j. suci , percy h. tannenbaum. *American Journal of Sociology*, 63(5):550–551, 1958.

[67] T. Nakagawa. Efficient top-down btg parsing for machine translation preordering. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 208–218, 2015.

[68] Y. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Publishing Company, Incorporated, 1 edition, 2014.

[69] A. Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *Proceedings of the Twenty-first International Conference on Machine Learning*, ICML '04, pages 78–, New York, NY, USA, 2004. ACM.

[70] M. Nickel and D. Kiela. Poincaré embeddings for learning hierarchical representations. *CoRR*, abs/1705.08039, 2017.

[71] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich. A review of relational machine learning for knowledge graphs. *Proceedings of the IEEE*, 104(1):11–33, 2016.

[72] M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816, 2011.

[73] N. F. Noy, D. L. McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001.

[74] A. Pease, I. Niles, and J. Li. The suggested upper merged ontology: A large ontology for the semantic web and its applications. In *In Working Notes of the AAAI-2002 Workshop on Ontologies and the Semantic Web*, page 2002, 2002.

[75] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[76] R. Raina, A. Madhavan, and A. Y. Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 873–880, New York, NY, USA, 2009. ACM.

[77] V. Raunak. Effective dimensionality reduction for word embeddings. *CoRR*, abs/1708.03629, 2017.

[78] A. V. Renssen. Gellish: a generic extensible ontological language - design and application of a universal data structure. 2005.

[79] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[80] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA, 1986.

[81] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[82] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, Nov. 1975.

[83] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *CoRR*, abs/1701.06538, 2017.

[84] E. Shortliffe. *Computer-based medical consultations: MYCIN*, volume 2. Elsevier, 2012.

[85] Siddharth-agrawal. Implementation of neural tensor network using scipy and numpy. `https://github.com/siddharth-agrawal/Neural-Tensor-Network`, 2014. [Online; accessed July-2018].

[86] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934, 2013.

[87] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[88] D. Steinkraus, I. Buck, and P. Y. Simard. Using gpus for machine learning algorithms. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 1115–1120 Vol. 2, Aug 2005.

[89] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.

[90] R. S. Sutton and A. G. Barto. *Reinforcement Learning : An Introduction*. MIT Press, 1998.

[91] A. Swartz. Musicbrainz: A semantic web service. *IEEE Intelligent Systems*, 17(1):76–77, 2002.

[92] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[93] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, and M. Gamon. Representing text for joint embedding of text and knowledge bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, 2015.

[94] D. Vrandečić and M. Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57:78–85, 2014.

[95] D. Vrandečić and M. Krötzsch. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85, Sept. 2014.

[96] Q. Wang, B. Wang, and L. Guo. Knowledge base completion using embeddings and rules. In *IJCAI*, pages 1859–1866, 2015.

[97] R. West, E. Gabrilovich, K. Murphy, S. Sun, R. Gupta, and D. Lin. Knowledge base completion via search-based question answering. In *Proceedings of the 23rd international conference on World wide web*, pages 515–526. ACM, 2014.

[98] Wikipedia contributors. Plagiarism — Wikipedia, the free encyclopedia, 2004. [Online; accessed July-2018].

[99] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[100] J. Yang, Y. Zhang, and F. Dong. Neural word segmentation with rich pretraining. *CoRR*, abs/1704.08960, 2017.

[101] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

[102] M. Zhang, Y. Zhang, and G. Fu. Transition-based neural word segmentation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 421–431, 2016.

[103] Z. Zhong, L. Jin, S. Zhang, and Z. Feng. Deeptext: A unified framework for text proposal generation and text detection in natural images. *CoRR*, abs/1605.07314, 2016.

A. ANEXO A. Implementación del modelo Neural Tensor Network en Pytorch

```
import numpy as np
import torch as t
import torch.nn as nn
from torch.autograd import Variable
import external_funcs

class NeuralTensorLayer(nn.Module):
    def __init__(self,I_dim,S_dim,num_relations,embedding_matrix,
        ont_info=False ,ont_matrix=None,train_embeddings=False):
        #1. Initialization of the hyperparameters of the layer
        self.output_dim=S_dim #Number of slices
        self.input_dim=I_dim #Dimension of the embedding vectors
        self.ont_info=False #Is there ontological information?
        self.train_embeddings = train_embeddings #Are the
        #embeddings of the entities going to be trained?
        self.relations = num_relations #Number of relations
        if ont_info:
            self.ont_info=True
            self.input_dim=2*I_dim
            self.ont_matrix=ont_matrix

        #2. Initialization of the layer according to Pytorch
        super(NeuralTensorLayer, self).__init__()

        #3. Especification of the parameters of the layer
        self.W = nn.Parameter(nn.init.xavier_uniform(
            t.zeros((self.input_dim,self.input_dim,
                self.output_dim,num_relations))))
                #Dimension (D,D,K,R)

        self.V= nn.Parameter(nn.init.xavier_uniform(
            t.zeros((self.output_dim,2*self.input_dim
                ,num_relations)))) #Dimension (Kx2D,R)

        self.b= nn.Parameter(t.zeros((
            self.output_dim,1,num_relations)))
            #Dimension (K,1,R)

        self.U= nn.Parameter(t.ones((
            1,self.output_dim,num_relations)))
            #Dimension (1,K,R)
```

```
        #Matrix E (optional training): Dimension (N_entities, D)
        if self.train_embeddings:
            self.E=nn.Parameter(t.Tensor(embedding_matrix))
        else:
            self.E=embedding_matrix

    def forward(self, batch,eval=False):
        relations_batch = external_funcs.split_batch(batch, self.relations)
        #Split the batch of triplets into R sub-batches
        #(one for each relation) and index them by relation.
        predictions=[]
        k = self.output_dim
        if self.train_embeddings:
            entEmbed=t.Tensor.numpy(self.E.data)
        else:
            entEmbed=self.E

        for r in range(self.relations):
            training_data_batch=relations_batch[r]
            #Retrieve the batch corresponding to the current relation

            #TRAINING PASS:
            if not eval:
                #Gather all indexes for the different entities
                e1 = [x[0] for x in training_data_batch] #e1 pos and neg
                e2 = [x[1] for x in training_data_batch] #e2 pos
                e3 = [x[2] for x in training_data_batch] #e2 neg
                #Obtain the embeddings associated to the entities
                if self.ont_info:
                    e1_o_info=np.array([self.ont_matrix[x] for x in e1])
                    e2_o_info=np.array([self.ont_matrix[x] for x in e2])
                    e3_o_info=np.array([self.ont_matrix[x] for x in e3])
                    e1_input=Variable(t.Tensor(np.concatenate(
                        (e1_o_info,entEmbed[e1]),axis=1)))
                    e2_input = Variable(t.Tensor(np.concatenate(
                        (e2_o_info,entEmbed[e2]), axis=1)))
                    e3_input = Variable(t.Tensor(np.concatenate(
                        (e3_o_info,entEmbed[e3]), axis=1)))
                else:
                    e1_input = Variable(t.Tensor(entEmbed[e1]))
                    e2_input = Variable(t.Tensor(entEmbed[e2]))
                    e3_input = Variable(t.Tensor(entEmbed[e3]))
                #Generate positive and negative scores
```

```
            e1_pos=t.transpose(e1_input.data,0,1)
            e2_pos = t.transpose(e2_input.data, 0, 1)
            e1_neg = t.transpose(e1_input.data, 0, 1)
            e2_neg = t.transpose(e3_input.data, 0, 1)

            ff_product_pos=self.V.data[:,:,r].mm(
                t.cat([e1_pos,e2_pos],dim=0))
            ff_product_neg = self.V.data[:,:,r].mm(
                t.cat([e1_neg, e2_neg], dim=0))
            bilinear_products_pos=[]
            bilinear_products_neg=[]
            for i in range(k):
                btp_pos=t.sum((t.transpose(e1_pos,0,1).mm(
                    self.W.data[:,:,i,r])).mm(e2_pos),dim=0)
                btp_pos=btp_pos.unsqueeze(0)
                bilinear_products_pos.append(btp_pos)

                btp_neg = t.sum((t.transpose(e1_neg, 0, 1).mm(
                    self.W.data[:,:,i,r])).mm(e2_neg), dim=0)
                btp_neg = btp_neg.unsqueeze(0)
                bilinear_products_neg.append(btp_neg)

            concat_pos=t.cat(bilinear_products_pos,dim=0)
            concat_neg = t.cat(bilinear_products_neg, dim=0)

            pre_pos=concat_pos+ff_product_pos+self.b.data[:,:,r]
            pos_activation=t.tanh(pre_pos)
            pre_neg=concat_neg+ff_product_neg+self.b.data[:,:,r]
            neg_activation=t.tanh(pre_neg)

            score_pos=self.U.data[:,:,r].mm(pos_activation)
            score_neg=self.U.data[:,:,r].mm(neg_activation)

            predictions.append(t.cat(
                (score_pos,score_neg),dim=0))
        else:
        #VALIDATION / EVALUATION FORWARD PASS:
            if len(training_data_batch) == 0:
                predictions.append(None)
            else:
                #Gather indexes for all entities
                e1 = [x[0] for x in training_data_batch]
                e2 = [x[1] for x in training_data_batch]
                #Gather the label for each triplet
```

```
                labels = [x[2] for x in training_data_batch]
                labels=t.Tensor(labels)
                labels=labels.unsqueeze(0)
                if self.ont_info:
                    e1_o_info = np.array([self.ont_matrix[x] for x in e1]
                    e2_o_info = np.array([self.ont_matrix[x] for x in e2]
                    e1_input = t.Tensor(np.concatenate(
                        (e1_o_info, entEmbed[e1]), axis=1))
                    e2_input = t.Tensor(np.concatenate(
                        (e2_o_info, entEmbed[e2]), axis=1))
                else:
                    e1_input = t.Tensor(entEmbed[e1])
                    e2_input = t.Tensor(entEmbed[e2])

                e1_input = t.transpose(e1_input, 0, 1)
                e2_input = t.transpose(e2_input, 0, 1)
                ff_product = self.V.data[:, :, r].mm(t.cat(
                    [e1_input, e2_input], dim=0))
                bilinear_products = []
                for i in range(k):
                    c = self.W.data[:, :, i, r]
                    b = t.transpose(e1_input, 0, 1).mm(c)
                    a = b.mm(e2_input)
                    btp = t.sum(a, dim=0)
                    btp = btp.unsqueeze(0)
                    bilinear_products.append(btp)
                concat = t.cat(bilinear_products, dim=0)
                pre_pos = concat + ff_product + self.b.data[:, :, r]
                pos_activation = t.tanh(pre_pos)
                score_pos = self.U.data[:, :, r].mm(pos_activation)
                predictions.append(t.cat((score_pos, labels), dim=0))
        if not eval:
            #Scores for positive and negative examples
            predictions=t.cat(predictions,dim=1)
            return Variable(predictions)
        else:
            #Scores and labels
            return predictions
```

B. ANEXO B. Implementacion de la función de pérdida del modelo Neural Tensor Network en Pytorch

```
import torch as t
import torch.nn as nn

class Tensor_Loss(t.nn.Module):
    def __init__(self):
        super(Tensor_Loss, self).__init__()

    def forward(self, predictions, regularization, parameters):
        scalar = t.Tensor([0])
        aux = 1-predictions[0]+predictions[1]
        tmp1 = t.max(scalar.expand_as(aux), aux.data)
        tmp1 = t.sum(tmp1)
        tmp2 = t.sqrt(sum([t.sum(var**2) for var in parameters]))
        loss = tmp1 + (regularization * tmp2)
        return loss
```