Universidad de la República
Facultad de Ingeniería

Detección de antonimia en español con redes neuronales parasiamesas

Memoria de proyecto presentada a la Facultad de Ingeniería de la Universidad de la República por

Juan Camacho, Juan Cámera

en cumplimiento parcial de los requerimientos para la obtención del título de Ingeniero en Computación.

Tutor

Mathias Etcheverry ..... Universidad de la República

Tribunal

Libertad Tansini ..... Universidad de la República
Mercedes Marzoa ..... Universidad de la República
Luis Chiruzzo ..... Universidad de la República

Montevideo
martes 20 de setiembre, 2022

Detección de antonimia en español con redes neuronales parasiamesas, Juan Camacho, Juan Cámera.

Esta tesis fue preparada en LaTeX usando la clase iietesis (v1.1).
Contiene un total de 78 páginas.
Compilada el domingo 6 noviembre, 2022.
http://fing.edu.uy/es/inco

Resumen

La discriminación entre antónimos y sinónimos (ASD) es una tarea del área de PLN que consiste en determinar si un par de palabras es antónimos o sinónimos entre si. Esta tarea puede presentar una gran complejidad a la hora de realizarse de manera automática, ya que una palabra y su antónimo suelen aparecer en contextos similares, lo que por ejemplo, podría derivar en errores en un sistema de implicancia textual.

Este problema ha sido abordado en varios trabajos en los cuales se plantearon diferentes modelos para resolver el problema. Estos trabajos han sido enfocados principalmente para el idioma inglés, no existiendo, según nuestras investigaciones, un trabajo que aborde el problema para el español, así como tampoco la existencia de un dataset para ASD que sea en español. A partir de esto se plantea como objetivo abordar el problema en nuestro idioma mediante la creación de un dataset y la evaluación del modelo Parasiamesa presentado en Etcheverry and Wonsever (2019) utilizando el dataset creado.

El dataset fue generado mediante la consulta WordNet en español (Fernández-Montraveta et al., 2008) y tres diccionarios web. De cada una de las fuentes se extrajeron palabras con sus sinónimos y antónimos generando tuplas por cada una de ellas. Posteriormente se realizó un análisis de calidad del mismo la cual consistió en realizar una anotación manual de 200 tuplas. Se evaluó la concordancia entre anotadores obteniendo un valor 0.899 de la medida *Kappa* de Cohen (1960) así como la *accuracy* entre los anotadores y el dataset obteniendo valores de 0.9. Utilizando las tuplas se dataset se analizó las relaciones de antonimia y sinonimia generando el grafo de las relaciones, en donde se vio que los grafos poseen una gran componente conexa conteniendo la mayoría de las palabras, lo que indica una gran conectividad con la existencia de camino entre la mayoría de las palabras.

El particionamiento del dataset en entrenamiento, validación y *test* se realizó de tres maneras distintas, donde dos de ellas poseen separación léxica entre los conjuntos. En partición sin separación léxica se utilizó un particionamiento aleatorio estratificado, mientras que para las particiones con separación léxica se utilizó un algoritmo basado en el presentado en Shwartz et al. (2016) y un algoritmo diseñado a partir de las características observadas al analizar los grafos de las relaciones. Adicionalmente se realizó la simetrización de las tuplas de los conjuntos

obteniendo dos variantes para cada uno.

Para la evaluación de la red Parasiamesa se realizó nuestra propia implementación del modelo así como la implementación de dos modelos más a ser comparados. Estos modelos son una red neuronal de tipo *feed-fordward* completamente conectado y una red Siamesa. Para cada combinación de particionamiento y modelo se realizó un *Random Search* para buscar los hiperparámetros que mejor desempeño den a los modelos.

Finalmente se utilizaron los mejores modelos encontrados y se evaluaron utilizando los conjuntos de *test*. Utilizando la medida F1 pudo observarse que el modelo Parasiamesa en su versión preentrenada fue el modelo que obtuvo el mejor desempeño en la mayoría de los particionamientos obteniendo una medida F1 de hasta 0.9 para el particionamiento *Random*.

Tabla de contenidos

Resumen I

1. Introducción **1**
1.1. Cronograma 2

2. Conceptos previos **3**
2.1. Clasificación supervisada 3
2.1.1. Métodos de clasificación supervisada 4
2.1.1.1. Regresión logística 5
2.1.1.2. Redes neuronales 6
2.1.1.3. Redes siamesas 9
2.1.2. Metricas de evaluacion 10
2.1.3. Anotación 11
2.2. Word embeddings 12
2.2.1. FastText 14
2.3. Relaciones de semántica léxicas 14
2.3.1. Antonimia 15
2.3.2. Sinonimia 16
2.3.3. Otras relaciones léxicas 16
2.4. WordNet 16
2.4.1. WordNet en español 18

3. Discriminación entre Antónimos y Sinónimos **19**
3.1. Basado en patrones 19
3.2. Distribucionales 22
3.2.1. Parasiamesa 25

4. Creación del Dataset **29**
4.1. Obtención de datos 29
4.2. Confección del dataset 30
4.3. Análisis de calidad 31
4.4. Topología de la sinonimia y antonimia 32
4.5. Partición del dataset 34
4.5.1. Partición *Random* 34
4.5.2. Partición Léxica: Shwartz 35

4.5.3. Partición Léxica: Grafo 36

5. Experimentos **39**
5.1. Modelos 39
5.1.1. Base line: Concat 39
5.1.2. Siamesa 40
5.1.2.1. Siamesa ANT 41
5.1.2.2. Siamesa SYN 41
5.1.3. Parasiamesa 41
5.1.3.1. Parasiamesa pre-entrenada 42
5.2. Random Search 42
5.2.1. Análisis de las búsquedas 43
5.2.2. Mejores configuraciones 44
5.2.2.1. Análisis del entrenamiento 45
5.3. Análisis de resultados 46

6. Conclusión y trabajos futuros **51**

A. Partición léxica: Grafo **53**

B. Espacios de búsqueda del random search **57**

C. Gráficas de mejores modelos con simétricos **59**

Referencias **63**

Índice de tablas **67**

Índice de figuras **68**

Capítulo 1

Introducción

Este proyecto se desarrolla dentro del área del Aprendizaje Automático, más específicamente en el área de Procesamiento del Lenguaje Natural (PLN). Esta, se encarga del estudio del lenguaje, el significado de las palabras, la estructura de las oraciones, la traducción automática, entre otras tareas. Tradicionalmente se divide en diferentes etapas, entre las cuales se encuentran el análisis morfológico, análisis sintáctico y análisis semántico.

Dentro del análisis semántico, resulta de interés conocer las relaciones que existen entre las palabras como ser sinonimia, antonimia e hiponimia, entre otras. Distinguir estas relaciones entre las palabras es una tarea desafiante para abordar con métodos de aprendizaje automático.

La discriminación entre sinónimos y antónimos (ASD, por sus siglas en ingles) es una tarea de PLN que consiste en determinar si dos palabras son antónimos o sinónimos entre sí. Esta tarea, a pesar de parecer simple para las personas, es compleja de resolver automáticamente debido a la similitud que existe entre las representaciones utilizadas.

Los sinónimos hacen referencia a palabras que tienen significados iguales (o casi iguales), por ejemplo, frió y fresco, cálido y tibio, entre muchos otros. En cuanto a la antonimia, se definen a grandes rasgos como palabras que tienen significados opuestos, por ejemplo, frío y calor, alto y bajo, amigo y enemigo. A pesar de ser palabras con significados opuestos, son intercambiables manteniendo la validez sintáctica del enunciado. Esto quiere decir que una palabra puede ser reemplazada por un antónimo o un sinónimo obteniéndose una oración que sigue siendo del lenguaje. En otras palabras, antónimos y sinónimos pueden ocurrir en contextos similares, lo cuál puede llevar a errores graves en herramientas de procesamiento de lenguaje natural basadas en grandes colecciones de texto. Por ejemplo, un traductor automático que confunda al traducir una palabra por un antónimo estaría dando una traducción incorrecta, o un sistema de implicancia textual (*textual entailment*) que considere similares palabras que son opuestas.

ASD ha sido abordado en diferentes trabajos, generando distintos conjuntos de datos y estrategias para resolverlo principalmente en inglés Xie and Zeng (2021), Etcheverry and Wonsever (2019), Nguyen et al. (2017). Al momento de realizar este proyecto no existen, según nuestro conocimiento, trabajos sobre ASD para el español, ni existe un conjunto de datos para ASD en español.

El objetivo de este proyecto es la creación de un conjunto de datos para ASD en español, y la implementación de modelo de red Parasiamesa (Etcheverry and Wonsever, 2019) para resolver la tarea con el dataset creado. Se realizó un estudio exhaustivo de hiperparámetros así como variaciones del modelo utilizado.

La creación del conjunto de datos se realiza a partir de WordNet en español Fernández-Montraveta et al. (2008) y una selección de recursos *online*. Este es dividido en particiones para el entrenamiento, la validación y el test del modelo utilizado. Las particiones del conjunto de datos se realizan utilizando 3 diferentes algoritmos, una selección aleatoria, uno simple sin intersección léxica y uno sin intersección léxica basada en la estructura del grafo de la relación de antonimia.

A partir de este trabajo se escribió el artículo"Antonymy-Synonymy Discrimination in Spanish with a Parasiamese Network" el cual fue presentado y aceptado en la *Ibero-American Conference on Artificial Intelligence* 2022.

1.1. Cronograma

Este proyecto se desarrolló en un total de 16 meses, comenzando en abril de 2021 y finalizando en agosto de 2022. A continuación se presenta el detalle del trabajo realizado en los meses anteriormente mencionados.

- Mes 1: Estudio del estado del arte y relevamiento de recursos.
- Mes 2 y 3: Interacción con fuentes para la obtención de datos. Relevamiento de recursos y librerías. Definición de cuales se van a utilizar. Construcción del dataset.
- Mes 4 y 5: Ajustes al dataset y evaluación de la calidad. Fin de construcción del dataset.
- Mes 4 a 9: Implementación y experimentación de linea base, siamesa y parasiamesa con el dataset obtenido.
- Mes 9 a 12: Redacción del artículo.
- Mes 1 a 16: Documentación.

La planificación inicial del proyecto fue de 12 meses pero este plazo se vio extendido debido principalmente a la elaboración del artículo y demoras en los tiempos de ejecución de los experimentos.

Capítulo 2

Conceptos previos

En este capítulo mencionaremos conceptos que son de importancia para nuestro trabajo, tanto del ámbito lingüístico como del área de la inteligencia artificial y el procesamiento del lenguaje natural.

2.1. Clasificación supervisada

El aprendizaje automático (*Machine Learning*) es un área de la informática que tiene sus inicios en la década del 50 (TURING, 1950). Se enfoca en el desarrollo de técnicas y algoritmos capaces de aprender a partir de un cierto conjunto de datos, es decir, estos algoritmos modifican su comportamiento en base a los datos provistos. A los datos utilizados para que un algoritmo aprenda se les conoce comúnmente como conjunto de entrenamiento.

Una tarea habitual es utilizar un algoritmo para aprender a predecir una clasificación de los datos. Por ejemplo, el conjunto de datos podría consistir de una lista de atributos de una planta (altura, color, etc), y se desea predecir si es venenosa o no (Mitchell, 1997). Hay un gran numero de técnicas y algoritmos para realizar esta tarea, estos se dividen en dos grandes ramas conocidas como clasificación supervisada y clasificación no supervisada.

La clasificación no supervisada, es la rama del aprendizaje automático que utiliza conjuntos de entrenamiento no etiquetados. Esto quiere decir que a priori no se conoce su clasificación. En el ejemplo propuesto anteriormente, sería tener la lista de plantas y no saber cuáles son venenosas. En esta rama se utilizan principalmente algoritmos de *clustering*, aprendizaje de representación, reducción de dimensiones, *transfer learning*, entre otros. Los algoritmos de *clustering* buscan generar grupos entre los datos por el relacionamiento que existe entre estos. El aprendizaje de representaciones es una forma de aprendizaje que permite a un sistema aprender representaciones de los datos de forma que estos puedan ser utilizados en alguna tarea específica. La reducción de dimensiones tiene como objetivo reducir la dimensión de los elementos en representaciones vectoriales preservando propiedades

del espacio original.

En cuanto a la clasificación supervisada las categorías en las cuales se puede clasificar un elemento son conocidas, y los conjuntos de entrenamiento contienen la clasificación que le corresponde a cada elemento. En esta rama del aprendizaje automático existen diversos métodos, y estos utilizan los datos de diferentes formas para aprender a clasificarlos. En la siguiente sección se comentan algunos de los métodos de clasificación supervisada.

2.1.1. Métodos de clasificación supervisada

Existen diversos métodos para la clasificación supervisada basados en diferentes teorías matemáticas. Uno de estos es el clasificador bayesiano óptimo. Estos clasificadores se basan en el teorema de Bayes. El teorema de Bayes en su forma matemática es:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Donde:

- A y B son eventos.
- $P(A|B)$ es la probabilidad de que ocurra A dado que ocurrió B.
- $P(A)$ es la probabilidad de que ocurra A.
- $P(B)$ es la probabilidad de que ocurra B, esta debe ser distinta de cero.

El clasificador bayesiano óptimo considera el espacio de hipótesis H finito. Este busca maximizar la probabilidad de la hipótesis h dado qué se observó el conjunto de entrenamiento. En su forma matemática sería $P(h|D)$, dónde D son las instancias del conjunto de entrenamiento. Debido a que esta probabilidad resulta difícil de calcular se utiliza el teorema de Bayes para calcularla, utilizando conteo para estimar las demás probabilidades y la fórmula del teorema para calcular $P(h|D)$.

Existen otras formas de utilizar los datos de entrenamiento para aprender una clasificación. Otro método de clasificación es la de k vecinos más cercanos (*K-Nearest Neighbors* o KNN).

Este método consiste en, dado un elemento desconocido, seleccionar los K elementos más cercanos en el conjunto de entrenamiento. Una vez obtenidos estos K elementos, se utiliza su clasificación para clasificar al elemento desconocido. Una parte central de este método radica en definir una función de distancia entre los elementos.

Debido al gran aumento en el poder de cómputo que ha acontecido en los últimos años, métodos computacionalmente complejos han ganado popularidad. En la siguiente sección presentaremos el método de Regresión Logística y Redes Neuronales.

2.1.1.1. Regresión logística

Para un problema de clasificación binaria en la cual se quiere determinar si un vector pertenece a una determinada clase clase (C_1), la probabilidad de que un vector X pertenezca a la clase C_1 puede ser escrita como una función sigmoid (σ), con lo cual la formula de regresión logística es:

$$P(C_1|X) = \sigma\left(W^T X\right) = \frac{1}{1 + e^{-W^T X}}$$

Siendo W un vector de dimensión D de variables ajustables. Notar que $P(\neg C_1|X) = 1 - P(C_1|X)$. Donde la función σ es de la siguiente forma:

$$\sigma\left(t\right) = \frac{1}{1 + e^{-t}}$$

Esta función tiene la particularidad de que permite interpretar su salida como una probabilidad. Además es una función de clase C^∞ lo que quiere decir que es infinitamente derivable y todas sus derivadas son continuas. En la figura 2.1 puede observarse la gráfica de la función.

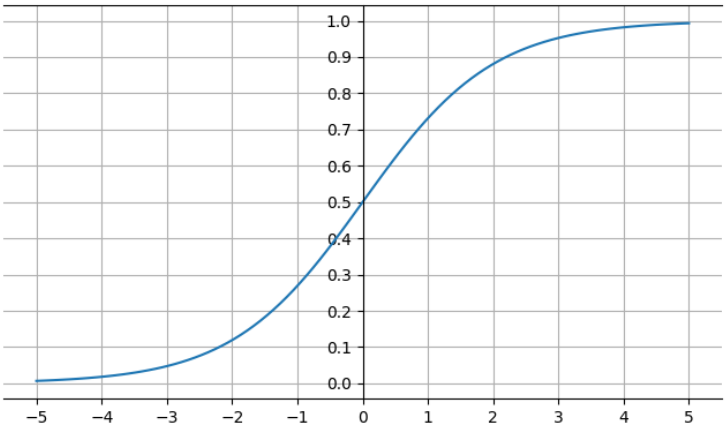

Figura 2.1: Gráfica de la función *sigmoid*

Dado que el vector W tiene la misma dimensión que el vector X, el modelo de regresión logística tiene tantas variables ajustables como elementos tenga el vector de entrada. Para determinar estos parámetros, se inicializan en valores aleatorios y

con los datos del conjunto de entrenamiento se realiza un descenso por gradiente, para de esta forma minimizar la función de pérdida, por ejemplo la función de entropía cruzada. En los caso donde la clasificación es multiclase en lugar de la función sigmoid se utiliza la función softmax, la cual dado un vector $z \in \mathbb{R}^K$ (Bishop, 2006):

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \; for \; i = 1...K$$

Esto genera que cada elemento de z pertenezca al intervalo $(0, 1)$ adicionalmente la sumatoria de todos los elementos de z da 1. Esto permite interpretar al resultado como una probabilidad.

2.1.1.2. Redes neuronales

Las redes neuronales artificiales, abreviado como ANN o NN por su sigla en inglés (*Artificial Neural Networks* o *Neural Networks*), es un área que contempla una familia de métodos de aprendizaje automático y se encuentran entre los más efectivos que se conocen actualmente. Este tipo de modelos se encuentran dentro del estado del arte para problemas como reconocer caracteres manuscritos, reconocimiento de voz, detección de huellas digitales o el reconocimiento facial, representaciones distribuidas de palabras, entre otros. (Mitchell, 1997)

El estudio de las redes neuronales artificiales se inspiró inicialmente en la observación de los sistemas de aprendizaje biológicos, que se construyen a partir de complejas redes de neuronas interconectadas. La construcción de las redes neuronales artificiales se realiza mediante una analogía aproximada de los sistemas biológicos, siendo constituidas por un conjunto interconectado de unidades simples, donde cada unidad toma un número de entradas y produce solo una salida. Los valores de entrada de una unidad, son valores salida de otras unidades, generándose de esta forma la interconexión (ejemplo red neuronal feed forward 2.4).

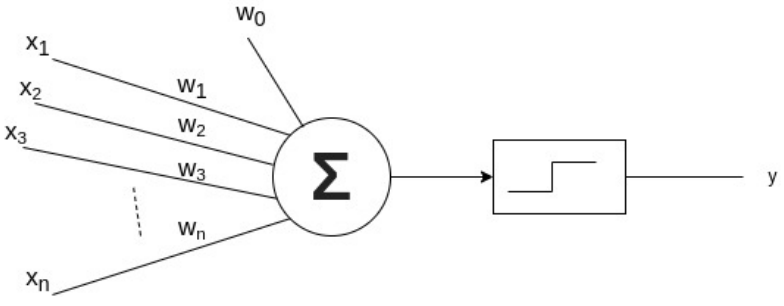

Figura 2.2: Ejemplo de neurona artificial.

Las unidades de las redes neuronales son llamadas neuronas. Una neurona, como puede verse en la figura 2.2, tiene como entrada un vector de valores reales, del

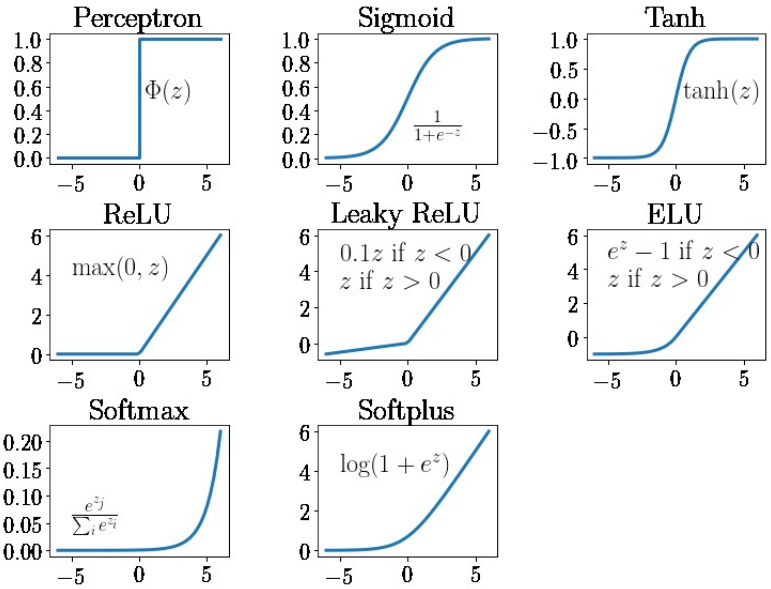

Figura 2.3: Funciones de activación populares.[1]

cuál calcula una combinación lineal de sus valores, y al resultado se le aplica una función no lineal llamada *función de activación* (figura 2.3). La salida de la neurona es el resultado de esta función. Dada una entrada $x_1, ..., x_n$ la combinación lineal aplicada por una neurona es una suma ponderada de forma $w_0 + x_1w_1 + x_2w_2 + ... + x_nw_n$ donde cada w_i es un valor real denominado peso. Al valor $-w_0$ se le denomina término independiente.

Los pesos w_i deben ser aprendidos para que el valor resultado sea el esperado. Una de las formas de aprender estos valores es comenzar utilizando valores aleatorios y de forma iterativa aplicar la neurona al conjunto de entrenamiento, ajustándolos cada vez que no da el valor esperado. Esto debe repetirse tantas veces como sea necesario hasta que la neurona clasifique todos los ejemplos de entrenamiento correctamente. Para el caso de una única neurona los pesos se ajustan realizando la siguiente asignación:

$$w_i = w_i + \Delta w_i$$

donde

$$\Delta w_i = \eta(t - o)x_i$$

Siendo η una tasa de aprendizaje, que se utiliza para controlar qué tan grande es el ajuste de los pesos, t es el valor esperado como resultado de la neurona y o el resultado provisto por ella. Este método es útil para casos en los cuales los

[1]Imagen extraída de `https://www.researchgate.net/figure/Common-activation-functions-in-artificial-neural-networks-NNs-that-introduce_fig7_341310767`

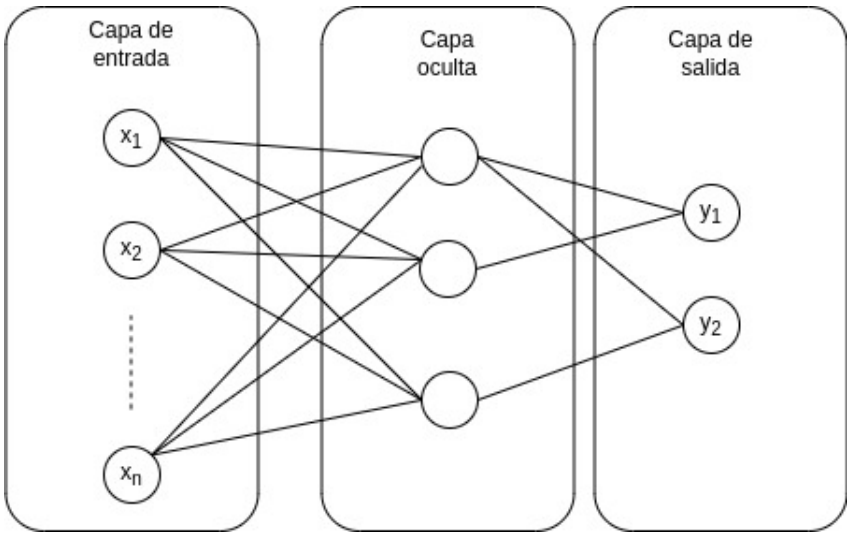

Figura 2.4: Red neuronal multicapa feed-fordward

ejemplos de entrenamiento son linealmente separables, ya que de lo contrario el método podría no converger.

Para el caso en que los espacios no pueden separarse linealmente, existe otro método de entrenamiento que se basa en realizar un descenso por gradiente en el espacio de posibles pesos. Este método es utilizado como base para el algoritmo de *backpropagation*, el cual se utiliza para el aprendizaje en redes de muchas unidades.

En los casos en los que se desea aprender un función no lineal, se deben utilizar redes neuronales multicapas, ya que una neurona sola no es útil en estos casos. Un esquema de una red neuronal multicapas básico se muestra en la figura 2.4, denominada red *feed-forward*. Este tipo de redes siempre posee una capa de entrada, una o varias capas ocultas y una capa de salida. Para el entrenamiento se utiliza el algoritmo de *backpropagation* (Rumelhart et al., 1986). Este algoritmo se basa en emplear un descenso de gradiente para intentar minimizar el error cuadrático entre los valores de salida de la red y los valores objetivo para las entradas (Rumelhart et al., 1986). Existen diferentes métodos para el ajuste de pesos inspirados en descenso por gradiente, uno de estos método es el algoritmo Adam (Kingma and Ba, 2014) comúnmente utilizado en la actualidad. Existen dos variantes extremas para el uso del descenso por gradiente. El descendo por gradiente estocástico el cual implica aplicar *backpropagation* para cada entrada del conjunto de entrenamiento, sin embargo esto resulta muy costoso. El otro extremo es total, en el cual se realiza la *backpropagation* una vez procesados todos los elementos. Generalmente se utiliza un punto medio de ambos enfoques en el cual se procesa en lotes de ejemplos, realizando una única *backpropagation* por cada lote. A la cantidad de ejemplos perteneciente a un lote se la conoce como *batch size*.

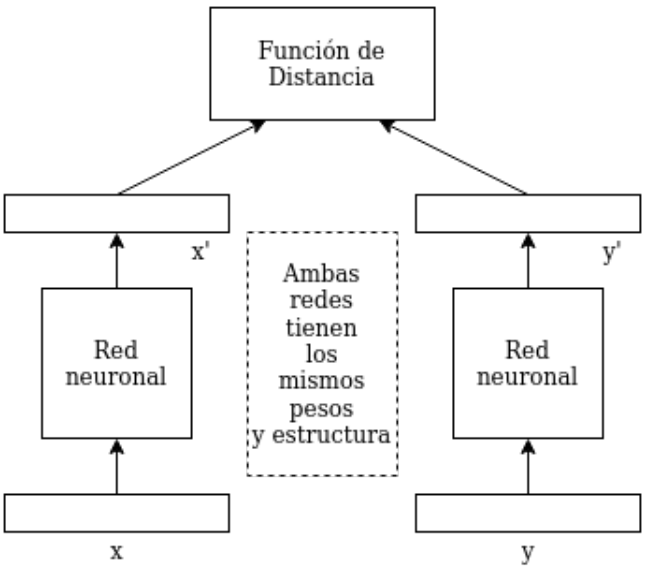

Figura 2.5: Red siamesa

2.1.1.3. Redes siamesas

Las redes siamesas constan de dos entradas y una red neuronal interna, la cual es ejecutada de forma independiente sobre cada vector de entrada. El valor resultante de una red siamesa está dado por una función de distancia que es aplicada a los vectores retornados por la red interna. Lleva el nombre de siamesa debido a que las redes neuronales aplicadas a sus entradas son idénticas en cuanto a la estructura y a los pesos. La estructura interna de las redes siamesas se encuentra ilustrada en la figura 2.5

En este tipo de redes se puede utilizar la función de pérdida *contrastive loss* (Khosla et al., 2020). Para esto se utilizan dos márgenes que denominamos margen de aceptación y margen de rechazo, donde $margen\_aceptacion <= margen\_rechazo$. La pérdida contrastiva busca obtener un valor menor que el margen de aceptación para vectores de la misma clase mientras que para vectores de clases distintas los vectores resultantes se busca que estén a una distancia mayor que el margen rechazo. Es decir se intenta acercar a los elementos que pertenecen a la misma clase y alejar a aquellos de clases distintas. Siendo d una funcion de distancia entre vectores e y un valor binario que indica si el par pertenece a la misma clase (1) o no (0), la función de pérdida contrastiva tiene la siguiente forma:

$$y \times max(dist - margen\_aceptacion, 0) + ((1-y) \times max(margen\_rechazo - dist, 0))$$

Siendo $dist = d(F(x_1), F(x_2))$ donde F es una red neuronal $\mathbb{R}^n \to \mathbb{R}^m$.

2.1.2. Metricas de evaluacion

Al tener ejemplos previamente clasificados es posible evaluar el desempeño de un método o algoritmo. De esta forma, se puede determinar qué tan bueno es para el problema de clasificación en cuestión o ajustarlo para obtener un modelo más acorde. Realizar un entrenamiento y posterior evaluación sobre un mismo conjunto de datos puede generar un sobreajuste del modelo, por lo que el conjunto de datos suele dividirse en tres subconjuntos: entrenamiento, validación y *test*. El modelo se entrena utilizando los ejemplos del conjunto de entrenamiento y se utiliza el conjunto de validación para evaluar el desempeño del clasificador hasta el momento. El modelo es entrenado varias veces y en cada iteración su desempeño se evalúa contra el conjunto de validación. Este procedimiento se realiza una cantidad n de veces, hasta que se determina que se encontraron los mejores hiperparametros. Luego se utiliza el conjunto de test para evaluar el desempeño del modelo final frente a datos no vistos anteriormente.

El desempeño es evaluado utilizando un conjunto de métricas. Cada una de ellas aporta diferente información sobre la clasificación predicha contra la clasificación correcta. Para poder utilizar las métricas es necesario definir los siguientes conceptos:

- Verdaderos positivos (V_p): Es la cantidad de elementos clasificados como positivos que efectivamente eran positivos. Es decir los elementos clasificados correctamente como positivos por el clasificador.
- Falsos positivos (F_p): Es la cantidad de elementos clasificados como positivos que en realidad no eran positivos. Es decir los elementos que fueron erróneamente clasificados como positivos por el clasificador.
- Verdaderos negativos (V_n): Es la cantidad de elementos clasificados como negativos que efectivamente eran negativos. Es decir elementos clasificados correctamente como negativos por el clasificador.
- Falsos negativos (F_n): Es la cantidad de elementos clasificados como negativos que en eran positivos. Es decir la cantidad de elementos positivos que fueron erróneamente clasificados como negativos por el clasificador.

Exactitud La exactitud (*accuracy* en inglés) mide el porcentaje de aciertos del clasificador. Esta métrica, al no diferenciar en clases, puede ser problemática cuando el porcentaje de elementos positivos es muy bajo. Por ejemplo, si en un dataset hay solamente un 10 % de elementos positivos y un clasificador clasifica a todos los elementos como negativos, tendría un 90 % de exactitud. La fórmula para esta métrica está dada por:

$$Exactitud = \frac{V_p + V_n}{V_p + V_n + F_p + F_n}$$

Precisión La precisión mide dentro de los elementos clasificados como positivos la exactitud de los mismos. Se la puede ver como una exactitud acotada a los elementos que recibieron una clasificación positiva. Cuando un clasificador tiene una precisión de X %, y clasifica a una instancia como positiva, entonces esa instancia tiene un X % de ser verdaderamente positiva. La fórmula para está métrica está dada por:

$$Precisión = \frac{V_p}{V_p + F_n}$$

Recuperación La recuperación (*recall* en inglés) es una medida del porcentaje de elementos de la clase positiva que fueron identificados por el clasificador. Se la puede ver como el porcentaje de elementos positivos que fueron identificados como tales. Esta métrica ayuda a identificar qué tantos elementos positivos el clasificador está pasando por alto. Por ejemplo, un clasificador con una recuperación de 25 % solo es capaz de identificar al 25 % de todos los elementos positivos que le fueron suministrados. La fórmula de esta métrica está dada por:

$$Recuperación = \frac{V_p}{V_p + F_n}$$

Medida F_β La medida F_β es una métrica que combina a la precisión y a la recuperación. Estas dos medidas son combinadas debido a que no otorgan suficiente información para evaluar el desempeño de un clasificador de forma independiente. Esta métrica posee un parámetro β el cual se utiliza para indicar qué tan importante es la precisión con respecto de la recuperación. Este parámetro resulta importante para aquellas aplicaciones que por su contexto favorecen a una métrica por encima de la otra. Por ejemplo, en detección de transacciones fraudulentas es preferible tener una recuperación más alta a costa de bajar un poco la precisión. La fórmula de esta métrica está dada por:

$$F_\beta = \frac{(1 + \beta^2).precisión.recuperación}{\beta^2.precisión + recuperación}$$

Los valores de $\beta < 1$ generan que la recuperación sea más importante para el cálculo de la métrica. Por el contrario, valores de $\beta > 1$ generan que la precisión sea más importante. En los casos en que la precisión y recuperación sean igual de importantes se utiliza $\beta = 1$, por lo cual se le llama simplemente medida F y la fórmula se simplifica dando como resultado:

$$F_1 = \frac{2.precisión.recuperación}{precisión + recuperación}$$

2.1.3. Anotación

No siempre se cuenta con un conjunto de datos que contenga la clasificación de los elementos. En estos casos se suele realizar un proceso de anotación manual del

conjunto de datos.

El proceso de anotación de un conjunto de datos, es realizado por una o varias personas que revisan individualmente el conjunto de datos y manualmente asignan una clasificación a cada elemento. La dificultad de la anotación está en qué tan fácil o difícil sea para un individuo determinar la clasificación de los elementos del conjunto y en la cantidad de elementos que contenga.

Esta tarea posee una dificultad adicional, la propia subjetividad de los individuos que anotan el dataset. Este es un problema común cuando se trata de anotar conjuntos de datos relacionados al lenguaje, dado que personas diferentes pueden realizar diferentes interpretaciones de las palabras, y por lo tanto, llegar a conclusiones distintas.

Por ejemplo, si se desea determinar si *inhóspito* y *selvático* son antónimos o sinónimos, un anotador podría considerar que ambas palabras son sinónimos dado que ambos lugares no son habitables. Sin embargo, otro anotador podría considerarlas antónimos, ya que un lugar inhóspito suele estar desprovisto de vida y un lugar selvático esta lleno de vida. Se podría argumentar que una clasificación es predominante sobre la otra según el sentido en el que se usan más comúnmente esas palabras, pero ambas clasificaciones son igualmente válidas.

Para evaluar el acuerdo existe el concepto de concordancia entre anotadores. La concordancia hace referencia a qué tan de acuerdo están los anotadores entre sí en las clasificaciones realizadas. Una métrica habitualmente utilizada para medir concordancia es la *Cohen's Kappa* (Cohen, 1960). La fórmula de la métrica está dada por:

$$k = \frac{p_o - p_e}{1 - p_e}$$

Siendo p_o la probabilidad empírica de concordancia en la etiqueta asignada a cualquier ejemplo y p_e la concordancia esperada cuando ambos anotadores asignan etiquetas aleatoriamente.

2.2. Word embeddings

Los *Word embeddings* son un tipo de representación usado en el procesamiento del lenguaje natural. Consisten en la representación de una palabra a un vector de un espacio multidimensional usando grandes colecciones de texto.

Los *words embeddings* están basados en la hipótesis distribucional (Harris, 1954). Dicha hipótesis establece que palabras que aparecen en contextos similares tienen significados similares. Esta idea permite utilizar la distribución de las palabras en

un *corpus* para generar vectores en un espacio multidimensional, de forma tal que, palabras con distribuciones similares tengan vectores cercanos en el espacio.

Existen varios métodos para generar *words embeddings* a partir de un corpus o conjunto de documentos. Una forma de generar vectores es crear una matriz donde las columnas están dadas por las palabras del vocabulario y las filas por los documentos del corpus. La matriz puede ser poblada con diferentes valores como por ejemplo la frecuencia de la palabra en cada documento, o el *tf-idf* de la palabra. La métrica de *tf-idf* consta de realizar el producto de dos términos. El primero se denomina frecuencia de término, que es la cantidad de veces que aparece un término t en un documento d:

$$tf_{t,d} = count(t, d)$$

Usualmente se suele tomar log_{10} de la frecuencia ya que una palabra que aparece muchas veces no necesariamente la hace relevante en el significado del documento. Dado que no se puede tomar el logaritmo de 0 usualmente se aplica la siguiente fórmula:

$$tf_{t,d} = log_{10}(count(t, d) + 1)$$

El segundo factor utilizado en este método se utiliza para darle mayor peso a aquellas palabras que aparecen en unos pocos documentos. La frecuencia de documentos df_t es la cantidad de documentos en los que aparece el término t. El valor *idf* se define mediante la fracción N/df_t, donde N es la cantidad total de documentos. A esta medida también se le suele tomar el logaritmo, dándonos como resultado la siguiente fórmula:

$$idf_t = log_{10}\left(\frac{N}{df_t}\right)$$

Uniendo estas dos fórmulas obtenemos los valores $w_{t,d}$ para una palabra t en un documento d:

$$w_{t,d} = tf_{t,d} \times idf_t$$

Debido a que un corpus suele tener una gran cantidad de documentos los vectores generados de esta forma tienden a tener un gran numero de dimensiones (la cual depende de la cantidad de documentos). Por este motivo se suelen utilizar técnicas de reducción de dimensiones como el análisis de componente principal presentado en Hotelling (1936).

Otro método muy importante para la generación de *embeddings* es el método de Skip-gram con muestreo negativo. Los *embeddings* generados por Skip-Gram son *embeddings* estáticos, es decir que el método aprende una representación fija para cada elemento del vocabulario, por lo que una palabra tendrá siempre el mismo vector asociado independientemente del contexto en el cual aparezca.

Para generar los *embeddings* se entrena a un clasificador binario. El clasificador tiene la tarea de predecir si una palabra w_1 es probable que aparezca cerca de otra palabra w_2. El resultado de la tarea de clasificación no es relevante, lo que realmente importa en este método son los pesos que aprendió el clasificador, debido a que son los *embeddings*.

La idea detrás de Skip-Gram con muestreo negativo es la siguiente:

1. Se considera una palabra objetivo y un contexto de esta como ejemplos positivos.
2. Se muestrean aleatoriamente otras palabras del lexicón para ser utilizadas como ejemplos negativos.
3. Se entrena un clasificador logístico para clasificar a las dos clases.
4. Se utilizan los pesos obtenidos por el clasificador como el *embedding* de la palabra objetivo.

Los vectores generados por este método poseen varias ventajas con respecto a otros tipos de *embeddings*. La principal ventaja de esta representación con respecto a tf-idf, es que se generan vectores pequeños, densos y con entradas reales.

2.2.1. FastText

FastText es una implementación de *word embeddings* que utiliza representación de n-gramas para lograr una mejor representación de las palabras especialmente en lenguajes que poseen una gran cantidad de inflexiones. (Bojanowski et al., 2016)

FastText posee versiones pre-entrenadas para unos 158 idiomas. Para este trabajo se utiliza la versión pre-entrenada de FastText para español (Joulin et al., 2016), la cual fue entrenada utilizando la versión de Wikipedia en español.

2.3. Relaciones de semántica léxicas

Se entiende como relaciones de la semántica léxica, a las relaciones entre los significados de las palabras. Dentro de estas podemos encontrar la antonimia, la sinonimia y la hiperonimia-hiponimia, entre otras.

En esta sección nos vamos a centrar en explicar algunas de las relaciones léxicas existentes, dándole énfasis a la antonimia y a la sinonimia que son el foco de estudio de este trabajo.

2.3.1. Antonimia

La antonimia es una relación léxica en la cual los significados de las palabras son opuestos. Por ejemplo los pares de palabras frío y calor, grande y chico, alto y bajo, son antónimos.

Los antónimos se suelen dividir en diferentes categorías dependiendo de las características de las palabras. A continuación consideramos las tres más aceptadas y generalmente usadas:

- **Antónimos graduales o escalares**: Estos antónimos se caracterizan por ser elementos opuestos en una escala. Por ejemplo: frío y calor, fresco y tibio. Ambas parejas hacen referencia a la temperatura, pero de esta escala, a casos opuestos.
- **Antónimos polares**: Estos antónimos se caracterizan por no tener un punto intermedio. Esto implica que la negación de uno resulta en la afirmación del otro. Por ejemplo: prendido y apagado, par e impar, vida y muerte.
- **Antónimos recíprocos**: Estos antónimos se caracterizan porque las palabras que los conforman son recíprocas entre sí. Generalmente son los nombres de roles de una relación entre dos elementos. Por ejemplo: comprar y vender, dar y recibir, padre e hijo.

Cuando dos palabras se encuentran relacionadas por la antonimia, a pesar de tener significados perceptiblemente opuestos, comparten una porción importante de significado, siendo diferentes u opuestas solamente en una cierta dimensión del mismo y a esto se le denomina la paradoja de la simultánea igualdad diferencia, es decir las palabras antónimas son opuestas en algún sentido y a su vez similares. En el ejemplo frío y calor, ambas refieren a la temperatura pero son opuestas en la magnitud, por lo cual la oración “Hace calor afuera” es incompatible con la oración “Hace frío afuera”, solo una de estas puede ser verdadera en un momento y lugar dado.

En el estudio lingüístico de la relación de antonimia, se ha encontrado que algunos antónimos son mas representativos que otros. Por ejemplo, si se le pregunta a una persona un antónimo de “rápido” espontáneamente se obtiene como respuesta “lento”, sin embargo “despacio” también es una respuesta válida. Existen varios trabajos que tratan sobre las características que genera esta diferenciación (Paradis et al., 2009) (Paradis, 2010).

2.3.2. Sinonimia

La sinonimia es la relación léxica que vincula palabras cuyo significado es el mismo o similar para al menos una acepción o contexto. Puede decirse que dos palabras X e Y son sinónimos si existe un contexto tal que al sustituir X por Y en una oración no se altera la condición de verdad de la misma. Por ejemplo, si se sustituye la palabra frío por fresco en la oración "Afuera esta frío" el significado no cambiaria, ni se perdería el sentido. Otros ejemplos de pares de sinónimos pueden ser duro y rígido, blando y flácido, o tirar y arrojar. Según Cruse (2010), los sinónimos suelen tener un grado significativo de superposición semántica y un bajo grado de contrastividad implícita. Esto quiere decir que los significados de las palabras que son sinónimos se superponen, pero eso no basta para que dos palabras sean sinónimos. Por ejemplo *collie* y *beagle* son dos razas de perros, por lo tanto tienen superposición semántica, sin embargo no son sinónimos. Para que dos palabras sean sinónimos, deben tener un bajo grado de contrastividad.

En Cruse (2010) se menciona que la sinonimia puede dividirse en dos categorías. Por un lado, la sinonimia total o absoluta, que se da cuando dos palabras son intercambiables en todos los contextos, este tipo de sinonimia es muy poco frecuente. Luego tenemos a la sinonimia parcial que se da en la mayoría los casos, cuando existe algún contexto para el cual los términos son intercambiables.

2.3.3. Otras relaciones léxicas

Además de la antonimia y la sinonimia existen otras relaciones léxicas. Como la hiperonimia y la meronimia.

Hiperonimia La hiperonimia es una relación donde una palabra es más general y otra más especifica. En estos casos se dice que la palabra más general es hiperónimo de la más especifica. Por ejemplo, "ave" es hiperónimo de "gallina", "persona" es hiperónimo de "niño". En contraparte, una palabra es hipónimo de otra si es una especificación de la otra. Es decir "gallina" es un hipónimo de "ave' y "niño" es hipónimo de "persona". Cuando dos palabras tienen un hiperónimo en común, se les denomina cohipónimos. Por ejemplo "gallina" y "golondrina" son cohipónimos entre si porque ambas son hipónimos de ave.

Meronimia La meronimia es la relación léxica entre un par de palabras A y B donde A es merónimo de B si A forma parte de B, por ejemplo, dedo es merónimo de mano. Por otro lado, la relación inversa se denomina holonimia, siendo mano un holónimo de dedo. Cuando dos palabras A y B forman parte de un mismo C, se dice que son comerónimos.

2.4. WordNet

WordNet (Miller, 1995) es una base de datos léxica del idioma inglés, gratuita

Figura 2.6: Captura de pantalla de la web de WordNet con *synsets* para la palabra *hot*. [2]

y disponible públicamente, donde tanto adjetivos, sustantivos como verbos están agrupados en grupos llamados *synsets*. Los *synsets* agrupan varias palabras que son sinónimos y hacen referencia a un significado común, con lo cual distintos *synsets* refieren a conceptos distintos. WordNet además de ser una base de datos léxica es un tesauro, debido a que mantiene las relaciones léxicas entre las palabras. Algunas de las relaciones que se encuentran en WordNet son: sinonimia, antonimia, hiperonimia, hiponimia y homónimos.

Cada *synsets* se vincula con otros mediante un pequeño conjunto de relaciones conceptuales. Además, contiene una pequeña definición denominada *gloss*, y en la mayoría se encuentran uno o más ejemplos de oraciones que muestran el uso de las palabras miembros del mismo. Por otro lado, una palabra con varios significados distintos va a estar representada en tantos *synsets* como significados tenga, y de esta manera cada par significado - forma de palabra es único dentro de WordNet.

Una de las relaciones más frecuentes entre los *synsets* es la relación de hiperonimia - hiponimia, que une *synsets* más genéricos con *synsets* de significado más específico. WordNet distingue entre sustantivos comunes y nombres de personas o entidades geográficas. En la figura 2.6 se puede observar ejemplos de *synsets* en

[2]Captura de pantalla tomada de `http://wordnetweb.princeton.edu/perl/webwn?o2=&o0=1&o8=1&o1=1&o7=&o5=&o9=&o6=&o3=&o4=&s=hot&i=4&h=01100000000000000000000#c`

los cuales se encuentra la palabra *hot*.

Si bien WordNet se construyó para el idioma inglés, existen varios proyectos que se encargan de generar versiones en diferentes idiomas. Open Multilingual WordNet (OMW) es una gran base de datos multilenguaje que conecta distintos proyectos de distintos idiomas. La motivación de este proyecto es poder utilizar WordNet en diferentes idiomas de forma sencilla y actualmente conecta WordNets en más de 200 idiomas con la versión 3.0 de WordNet en inglés.

2.4.1. WordNet en español

WordNet en español es un recurso presentado en Fernández-Montraveta et al. (2008). Está compuesto por la traducción al español de los *synsets* que se encuentran presentes en WordNet en inglés, sumándosele un corpus anotado paralelo con la definición de cada *synsets*. En este recurso se mantiene la estructura lo más similar posible a la versión original en inglés.

Su trabajo consistió en la traducción de las variantes y las glosas al español, cambiando las notaciones en los casos en los que la categoría morfosintáctica no es la misma en ambos idiomas, realizando una alineación a nivel de palabras siempre que fuera posible. En algunos casos la equivalencia no es uno a uno, dado que puede haber distinta cantidad de sinónimos para un mismo concepto en cada idioma. Esto conlleva a que los *synsets* en español no tienen necesariamente la misma cantidad de variantes que en el inglés.

En este recurso se encuentran traducidas aproximadamente la mitad de las glosas presentes en WordNet al momento de publicado el trabajo de Fernández-Montraveta et al. (2008), lo cual quiere decir que aproximadamente unas 30.000 entradas léxicas (nominales y verbales) están disponibles en español. Al igual que WordNet en inglés, WordNet en español esta disponible públicamente y se puede utilizar de manera gratuita con fines de investigación.

Capítulo 3

Discriminación entre Antónimos y Sinónimos

La discriminación entre antónimos y sinónimos es la tarea encargada de distinguir cuando dos palabras son antónimos o sinónimos entre si. En este capítulo vamos a mencionar y analizar distintos métodos propuestos por distintos autores que se encargan de detectar relaciones léxicas, especialmente la discriminación de antónimos y sinónimos. Éstos métodos se separan en dos categorías: métodos basados en patrones y métodos distribucionales.

3.1. Basado en patrones

Dentro de los primeros trabajos realizados con modelos basados en patrones se encuentra el presentado por Hearst (1992). Este trabajo se enfoca en la detección de hipónimos en el idioma inglés a través de patrones que se repiten en sus apariciones dentro de los textos.

Algunos de los patrones utilizados por Hearst, traducidos al español, serían:

- **"tanto A como B y/o C"**, donde B y C serían hipónimos de A
- **"A, B u/y otro C"**, donde A y B serían hipónimos de C
- **"A incluyendo/especialmente B, C y/o D"** donde B, C y D serían hipónimos de A

Cuando un par de hipónimos es descubierto utilizando los patrones, la frase fue lematizada y tratado como una unidad atómica. El procedimiento utilizado se describe a continuación, a partir de los primeros patrones, los cuales fueron detectados mirando textos manualmente, se obtuvieron los primeros pares de hipónimos. Luego se buscó en el corpus las ocurrencias de los mismos en los casos donde aparecían uno cerca del otro, observando su entorno. A partir de estas observaciones se infieren nuevos patrones comunes que se repiten. Con estos patrones se pueden conseguir nuevos pares de palabras y repetir el procedimiento.

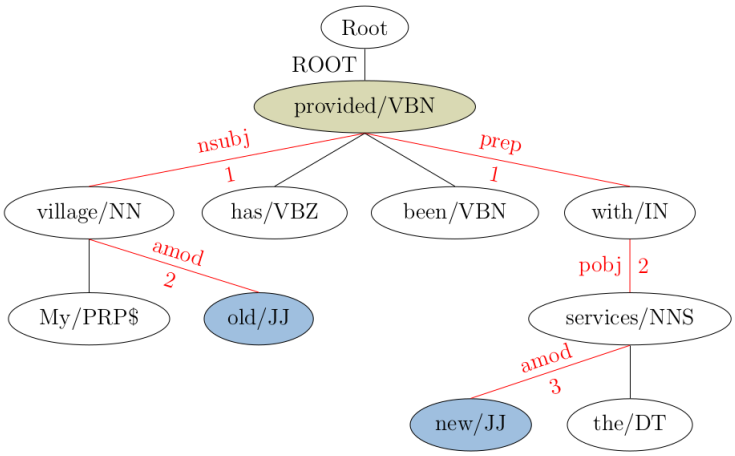

Figura 3.1: Árbol sintáctico de dependencias para la oración "My old village has been provided with the new services" donde el camino de "old" a "new" se encuentra marcado de color rojo y esta formado por (old-village-provided-with-services-new). Imagen extraída de Nguyen et al. (2017)

Posteriormente, Schulte im Walde and Köper (2013) propusieron un modelo para la distinción de antónimos, sinónimos e hiperónimos para el idioma alemán, representando vectorialmente cada par de palabras mediante un conjunto de patrones inferido a partir de pares de palabras de entrenamiento. Para realizar la clasificación de cada relación, calcularon el vector promedio de las palabras incluidas en el conjunto de entrenamiento (80 % del total) y predijeron la relación para los pares del conjunto de evaluación (20 % restante) clasificando cada par de prueba con el vector promedio más cercano según la distancia coseno. Mediante este método, en el conjunto de datos que generaron, obtuvieron para la distinción entre antónimos y sinónimos una F de 70,75.

En Nguyen et al. (2017) presentaron el model AntSynNET, que utiliza una red neuronal de tipo *long short-term memory (LSTM) network* (Hochreiter and Schmidhuber, 1997) para la detección de antónimos y sinónimos. Su método está basado en estudios basados en corpus sobre la antonimia, que sugieren que los opuestos coocurren con frecuencia, utilizando patrones como los principales indicadores de la co-ocurrencia de pares de palabras para generar distinción entre antónimos y sinónimos.

Dado que una oración puede ser representada mediante un árbol sintáctico, y que la teoría de grafos dice que cada vértice de un árbol contiene un único camino simple[1] hacia otro vértice, si se genera el árbol sintáctico de una oración se puede

[1]Un camino simple entre dos nodos es aquel camino que no repite vértices.

generar un único camino entre el par de palabras objetivo, el cual solo contiene las palabras de la oración relevantes a utilizar en el patrón. En la figura 3.1 se puede observar como ejemplo el árbol sintáctico de dependencias para la oración "*My old village has been provided with the new services*" (en español: mi viejo pueblo ha sido provisto con nuevos servicios). El patrón léxico-sintáctico para el par de antónimos viejo, nuevo se determina mediante el camino simple entre ambos lemas, marcado de color rojo.

Para la construcción de los patrones, cada nodo del grafo fue representado mediante un vector generado a partir de la concatenación de los valores que representan el lema de la palabra ($\overrightarrow{v}_{lemma}$), el *part-of-speach tag* ($\overrightarrow{v}_{pos}$), el nombre de la dependencia ($\overrightarrow{v}_{des}$), y la distancia entre la raíz del grafo y la palabra ($\overrightarrow{v}_{dist}$), siendo esta 0 cuando la palabra es la raíz. Con lo cual cada nodo que a ser utilizado en el patron esta representado por un vector de la siguiente forma:

$$\overrightarrow{v}_{nodo} = \overrightarrow{v}_{lemma} \oplus \overrightarrow{v}_{pos} \oplus \overrightarrow{v}_{des} \oplus \overrightarrow{v}_{dist}$$

Donde $\oplus$ denota a el operador de concatenación. Por otro lado, un patrón p construido a partir de una secuencia de nodos $n_1, ..., n_k$ se representa con la secuencia de vectores de cada nodo: $p = [\overrightarrow{n_1}, ..., \overrightarrow{n_k}]$. El vector del patrón ($v_p$) es obtenido al aplicar la red neuronal a la secuencia de vectores. Este modelo utiliza una red neuronal recurrente (RNN) de tipo LSTM. Dada una secuencia de palabras de entrada $p = [\overrightarrow{n_1}, ..., \overrightarrow{n_k}]$, la RNN procesa cada palabra por vez, manteniendo un estado interno que depende de las palabras procesadas anteriormente. El estado retornado al procesar el último nodo del patrón se considera como el vector que lo representa.

Los autores presentaron dos modelos para la distinción de antónimos y sinónimos, Pattern-based AntSynNET y Combined AntSynNET. En el primer modelo dado un par de palabras (x, y) se inducen patrones a partir de ellas utilizando un corpus, donde cada patrón representa el camino de x a y. Cada patrón es pasado por la red neuronal obteniendo un vector para cada uno de ellos, y luego, mediante la siguiente fórmula se obtiene el vector del par (x,y):

$$\overrightarrow{v}_{xy} = \frac{\sum_{p \epsilon P(x,y)} \overrightarrow{v_p} \cdot c_p}{\sum_{p \epsilon P(x,y)} c_p}$$

Donde P(x,y) es el conjunto de patrones de x hacia y, C_p es la frecuencia con la que ocurre ese patrón. Finalmente se utiliza una regresión logística para la clasificación del par, donde es clasificado como positivo si el resultado es mayor a 0.5. En la figura 3.2 se muestra un diagrama del funcionamiento del modelo.

El modelo fue entrenado con un conjunto de datos de sinónimos y antónimos previamente utilizado en Nguyen et al. (2016), donde los pares de datos fueron recolectados de WordNet y Wordlink. En el dataset se mantuvo una relacion 1:1

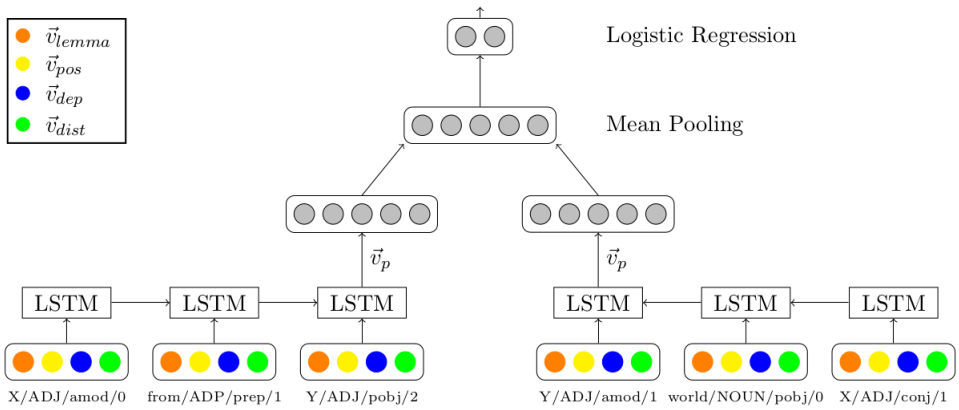

Figura 3.2: Imagen extraída de Nguyen et al. (2017) donde se ilustra el modelo AntSynNET presentado por los autores.

entre ambas clases y utilizaron la Wikipedia en inglés como corpus para la obtención de los patrones. Como resultado se obtuvieron medidas F de 0.776 para los adjetivos, 0.784 para los verbos, y 0.827 para los sustantivos.

El segundo modelo planteado será explicado en la próxima sección ya que tiene un enfoque combinado de patrones y distribucional.

3.2. Distribucionales

Los enfoques distribucionales buscan resolver el problema de ASD utilizando la distribución de los contextos de las palabras en un corpus. A diferencia de los enfoques basados en patrones, los cuales buscan relacionar palabras en un contexto mediante un patrón, los distribucionales consideran todos los contextos independientemente de la otras palabras. En la actualidad no se suele utilizar directamente la distribución de los contextos de las palabras en un corpus, se utiliza en forma implícita con los *word embeddings*.

Los sinónimos y antónimos de una palabra son generalmente intercambiables con la palabra en cuestión. Los antónimos y sinónimos son utilizados en contextos similares, lo que genera que el vector (o *embedding*) de una palabra dada tienda a estar cerca del los vectores de palabaras que son sinónimas o antónimas. Sin embargo, aunque los vectores sean cercanos, estos contienen suficiente información para lograr discriminar a los antónimos de los sinónimos. A continuación se presentan algunos modelos los cuales logran, a partir de *word embeddings* clasificar antónimos y sinónimos.

En el trabajo presentado en Vulić (2018) se utilizan *word embeddings* para generar una transformación de forma tal que ciertos vectores reducen su distancia

y otros vectores la aumentan. Este modelo, conocido como *ATTRACT-REPEL*, fue presentado originalmente en Mrkšić et al. (2017), el cual fue modificado para la tarea de ASD. Para lograr identificar la sinonimia y antonimia, este trabajo propone acercar a los sinónimos y alejar a los antónimos, con lo cual les impone la restricción *ATTRACT* a los sinónimos y la restricción *REPEL* a los antónimos.

Además de los elementos en el dataset de sinónimos (*syn*) y antónimos (*ant*), en Vulić (2018) se utiliza información adicional en forma de diferentes conjuntos, los cuales son:

- **Conjunto *antext*:** Estas tuplas son generadas de la siguiente manera: si dos palabras (w_i, w_j) son sinónimos y (w_i, w_k) son antónimos se agrega como antónimos en *antext* a (w_j, w_k). Este conjunto fue utilizado con la restricción *REPEL*
- **Conjunto *hip1*:** Este conjunto está conformado por hiperónimos extraídos de WordNet con el patrón IS-A. La idea detrás de este conjunto es que la hiperonimia suele ser confundida por sinonimia por los seres humanos, con lo cual agregar este conjunto puede ayudar a contrastar mejor las palabras. Este conjunto fue utilizado con la restricción *ATTRACT*.

Con esta información se evaluaron diferentes tareas de clasificación, cada una utilizando varias combinaciones de los conjuntos. La tarea relevante para este trabajo, de distinción de sinónimos y antónimos, obtuvo como mejores valores de medida F_1 0.969 para adjetivos, 0.975 para verbos y 0.879 para sustantivos en el mismo dataset utilizado en Nguyen et al. (2017). Estos valores fueron alcanzados utilizando los conjuntos de *syn* y *hip1* con restricción *ATTRACT* y el conjunto *ant* con restricción *REPEL*.

Ali et al. (2019) presenta un modelo en dos etapas. La primera etapa del modelo consiste en el entrenamiento del *DESTILLER*. Este se encarga de destilar representaciones específicas para la tarea. Esta primera etapa utiliza *word embeddings* preentrenados para el entrenamiento. Como resultado, el *DESTILLER* genera nuevas representaciones utilizando proyecciones no lineales de los vectores de entrada. Dichas proyecciones son generadas utilizando dos diferentes codificadores donde cada codificador consiste de una red neuronal *Feed-Forward* con dos capas ocultas y función de activación *Sigmoid*.

La segunda etapa consiste de un clasificador. En este trabajo se utiliza como clasificador a *XGBoost* (Chen and Guestrin, 2016). Como atributos del clasificador se utiliza:

- Valor de sinonimia y antonimia, el cual es calculado a partir de los vectores obtenidos en la fase 1.
- Valor de similitud distribucional, en este caso se usó el $cos(a, b)$ siendo a y b son los *word embeddings* preentrenados.

- Prefijos de negación. En este trabajo se consideran 14 prefijos de negación. Se agrega un atributo binario cuyo valor es 1 si alguna de las dos palabras candidatas difieren por uno de los prefijos de negación.

Este modelo es independiente de los *embedding* utilizados. A los efectos de mostrar esto los autores plantean las diferentes métricas con vectores aleatorios, *Glove*, *dLCE* y *ELMO*. El mejor valor de F1 presentado sin utilizar *dLCE* fue de 0.884, 0.891 y 0.884 para adjetivos, verbos y sustantivos respectivamente. Estas métricas fueron obtenidas en el mismo dataset utilizado por Nguyen et al. (2017).

Notar que este método además de los *word embeddings* se agregan algunos atributos extra para ayudar a la clasificación. Este acercamiento es común en otros trabajos, un ejemplo de este tipo de trabajos es Xie and Zeng (2021).

Xie and Zeng (2021) utiliza un modelo *Mixture of Experts* (Jacobs et al., 1991). Consiste en generar un cierto numero de redes neuronales, cada una es un experto. Cada experto toma los *word embeddings* (w_1, w_2), cuyo tamaño es d_e, y los proyecta a un subespacio de tamaño d_u, es decir, se transforman vectores de R^{d_e} en vectores de R^{d_u}. Dicha transformación da como resultado dos nuevos vectores denominados u_1 y u_2. Todos los expertos realizan proyecciones a espacios del mismo tamaño pero cada transformación es diferente para cada experto.

Una vez que cada experto generó los vectores u_1 y u_2 se genera un vector de atributos denominado r, el cual esta formado por $(u_1+u_2)\oplus|u_1-u_2|\oplus cos(u_1,u_2)\oplus f_{w_1,w_2}$, donde f_{w_1,w_2} es un atributo de negación idéntico al presentado anteriormente en Ali et al. (2019). Una vez generado el vector r cada experto calcula la probabilidad de que las palabras sean antónimas utilizando la función *sigmoid*. Utilizando este método, se obtuvieron como resultado Medida F_1 de 0.892 para adjetivos, 0.908 para verbos y 0.869 para sustantivos. Estas métricas fueron obtenidas en el mismo dataset utilizado por Nguyen et al. (2017).

También existen modelos que utilizan ambos enfoques en simultáneo, es decir, realizan una mezcla de patrones y *words embeddings*. Estos modelos son denominados como híbridos y un ejemplo de ellos es el segundo modelo presentado por Nguyen et al. (2017).

Este segundo modelo, denominado *Combined AntSynNET*, tiene en cuenta los patrones y la distribución de los pares para crear vectores combinados. Dado el par de palabras (x, y), la representación del vector combinado del par se determina usando la distribución de co-ocurrencia de las palabras y la ruta sintáctica del patrón, concatenando el vector de la palabra x, seguido del vector del patrón, y finalizando con el vector de y. Luego se utiliza la regresión logística, al igual que en el modelo puramente basado en patrones. Las medidas F reportadas por el autor para este modelo son de 0.784 para los adjetivos, 0.777 para los verbos y 0.855

para los sustantivos, superando al método basado en patrones puro en verbos y sustantivos.

3.2.1. Parasiamesa

El modelo de red parasiamesa fue presentado en Etcheverry and Wonsever (2019). Está inspirado en una red siamesa y busca explotar las características algebraicas de la relación de antonimia y sinonimia. En este trabajo se consideran las siguientes propiedades algebraicas para cada una de las relaciones:

Sinonimia:

1. **Reflexiva:** Todos las palabras son sinónimos de sí mismas, debido a que tienen el mismo significado.
2. **Simétrica:** Si A es sinónimo de B entonces B es sinónimo de A.
3. **Transitiva:** Si A es un sinónimo de B y B es un sinónimo de C entonces A y C son sinónimos.

Antonimia:

1. **Irreflexiva:** Una palabra no es antónima de sí misma.
2. **Simétrica:** Si A es antónimo de B entonces B es antónimo de A.
3. **Antitransitiva:** Si A es un antónimo de B y B es un antónimo de C entonces A y C no son antónimos. Debido a que tanto A como C tienen un significado opuesto a B, es probable que A y C sean sinónimos.

Estas propiedades son formuladas en Etcheverry and Wonsever (2019) bajo una determinada acepción de las palabras. Por ejemplo, en las tuplas de sinónimos (calor, bochorno), (bochorno, ardor) y (calor, ardor) podemos observar la transitividad; y si aplicamos el mismo razonamiento con las tuplas de antónimos (calor, frío) y (frío, bochorno) obtendríamos la tupla de sinónimos (calor, bochorno), lo que ejemplifica la antitransitividad de los antónimos.

Debido a que las palabras pueden tener varias acepciones, podrían darse casos en los cuales las propiedades anteriormente mencionadas no se cumplan, por ejemplo, al encontrar una palabra que pueda considerarse como antónima de sí misma, o como antónima de una palabra B si se considera una acepción y a su vez sinónima de la misma palabra B si se considera una acepción distinta.

La red siamesa, permite modelar las propiedades de la relación de sinonimia, dado que es adecuada para aprender una relación que es simétrica, transitiva y si se utiliza una función de clasificación apropiada, también genera una relación reflexiva.

En cambio, si se consideran las características de la relación de antonimia, una red siamesa no es adecuada para modelar la antiransitividad de la relación por la estructura que posee el modelo. Es por esta razón que en Etcheverry and Wonsever (2019) se presenta la red parasiamesa, la cual busca modelar la propiedad antitransitiva de la relación de antonimia.

La solución propuesta para el problema consta de aplicar dos veces la red neuronal interna en una de las ramas de la red siamesa como puede apreciarse en la figura 3.3. De esta forma el modelo se adecua para aprender a transformar los vectores de entrada, de forma tal que la distancia de los vectores resultantes es pequeña en caso de ser antónimos, y es mayor en caso de ser sinónimos.

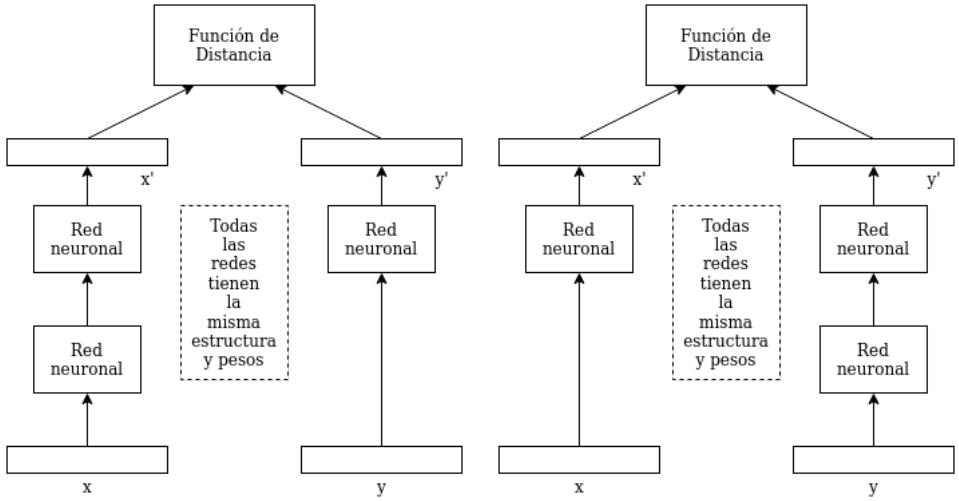

Figura 3.3: Estructura interna de una parasiamesa izquierda y derecha respectivamente, imagen basada en Etcheverry and Wonsever (2019)

Por definición la red parasiamesa no es simétrica. Para asegurar simetría ésta debe ser incluida mediante función de pérdida o mediante el conjunto de datos, es decir, que cada tupla del dataset tenga su contraparte simétrica.

Nótese que el modelo es no simétrico y existen dos posibles variaciones del mismo. Estas variaciones consisten en realizar la doble aplicación de la función sobre el lado derecho o sobre el lado izquierdo. En la figura 3.3 se observa la estructura de la red parasiamesa en sus versiones izquierda y derecha.

En Etcheverry and Wonsever (2019) también se propone una variante preentrenada de la parasiamesa. Basándose en la función de pérdida de la parasiamesa y en la propiedad 3 de la antonimia, si consideramos A antónimo de B, y B antónimo de C obtenemos las siguientes fórmulas:

$$F(A) = F(F(B))$$
$$F(F(B)) = F(C)$$

Donde F es la función de pérdida entrenada. Puede observarse que $F(A) = F(C)$ y considerando que A y C tienden a ser sinónimos, podríamos concluir en que F concuerda con la función resultante de la red siamesa. A partir de esta observación se plantea realizar un preentrenamiento de la red interna de la parasiamesa entrenándola en una red siamesa utilizando sinónimos como clase positiva.

Los mejores resultados alcanzados con este modelo, se dan para una red parasiamesa preentrenada utilizando como *word embeddings* FastText. Con esta configuración se alcanzaron valores de medida F_1 de 0.856 para adjetivos, 0.891 para verbos y 0.848 para sustantivos. Al momento de iniciar este proyecto estos resultados eran los mejores para la tarea de ASD. Actualmente existen modelos que supera estas métricas sin embargo el objetivo de este proyecto es la implementación y experimentación utilizando la red parasiamesa.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 4

Creación del Dataset

En este capítulo presentamos la construcción de un dataset de antónimos y sinónimos para el español. El mismo consiste en un conjunto tuplas de palabras que se encuentran relacionadas entre sí por una de estas dos relaciones léxicas. Se presenta el procedimiento realizado para la obtención, particionamiento y análisis sobre los datos obtenidos.

4.1. Obtención de datos

Las tuplas del dataset fueron generadas mediante la consulta de distintas fuentes o diccionarios en línea que se encuentran disponibles. Inicialmente se extrajeron los datos existentes en WordNet en español y luego se realizó un búsqueda de diccionarios existentes en línea, decidiendo utilizar los siguientes tres sitios:

- wordreference.com[1]
- antonimos.net[2]
- sinónimo.es[3]

Tanto en el primer sitio como en el tercero, para una palabra dada podemos encontrar sus sinónimos, antónimos y diferentes definiciones, mientras que en antonimos.net solamente se puede encontrar una lista de antónimos.

Como cada diccionario es un sitio web, se construyó un *scraper* para cada uno utilizando la librería Scrapy de Python (Kouzis-Loukas, 2016), y así poder consultar y extraer de forma automática cada palabra con sus respectivos antónimos y sinónimos. Todas las fuentes funcionan de la misma forma, a partir de una palabra se realiza la consulta y se obtiene la información, necesitando palabras a priori para realizar las consultas.

[1] https://www.wordreference.com/
[2] https://www.antonimos.net/
[3] https://www.xn–sinnimo-n0a.es/

Debido a la necesidad de tener un vocabulario inicial, utilizamos el conjunto de palabras presentes en WordNet en español (36681 palabras). Luego, mediante el *scraper* construido para ese sitio, se generó una tupla con cada palabra y las palabras indicadas en el sitio como antónimos o sinónimos. Si estas nuevas palabras descubiertas no se encontraban presentes en nuestro vocabulario, fueron agregadas al mismo, y luego utilizadas para obtener sus antónimos y sinónimos, aumentando el vocabulario de nuestro dataset.

Posteriormente, en busca de incrementar nuestro vocabulario, se decidió ejecutar los *scrapers* utilizando otro vocabulario inicial. Para esto se utilizó el corpus en español creado por Cardellino (2019), del cual se extrajo su vocabulario (3225303 palabras).

Una vez ejecutados todos los *scrapers*, obtuvimos dos datasets por cada recurso web (sinónimos.es, wordference.com y antonimos.es), uno generado utilizando como vocabulario inicial WordNet y otro utilizando Cardellino, los que se suman al generado completamente con WordNet, utilizando los antónimos y sinónimos que se encuentran presentes en éste.

4.2. Confección del dataset

El siguiente paso realizado fue la unificación de los datasets, para obtener una única fuente que contenga todas las tuplas anotadas. La unificación constó de dos etapas. La primera etapa fue unificar los datasets por un mismo vocabulario inicial, obteniendo entonces dos datasets. Al realizar esta unificación parcial notamos que a pesar de que el vocabulario inicial de Cardellino (3225303 palabras) es casi 88 veces mayor que el de WordNet (36681 palabras), la cantidad de tuplas obtenidas utilizando Cardellino (1105441 tuplas) no difiere radicalmente a las obtenidas a partir de WordNet (977408 tuplas). Encontramos que muchas palabras dentro del corpus de Cardellino no se encuentran en español, tienen errores ortográficos o son nombres propios, y generaron consultas sin resultados en nuestros recursos web ademas de la interconecctividad que dan estas relaciones, como se ve en la sección 4.4.

Luego de tener un dataset por vocabulario inicial, el siguiente paso fue unificar los dos datasets obtenidos. En el proceso de unificación, las tuplas repetidas, es decir tuplas donde tanto ambas palabras como la clasificación eran las mismas, fueron eliminadas. Dado que según nuestro criterio wordreference.com es la fuente más confiable, se tomó como base el dataset obtenido a partir de ésta.

Una vez unificadas las diferentes fuentes, las tuplas simétricas y reflexivas fueron eliminadas, ya que por un lado consideramos que las tuplas reflexivas no nos aportaban información relevante, y por el otro asumimos que ambas relaciones son simétricas. Esto nos permite simetrizar el dataset luego de haber generado los

conjuntos de entrenamiento, validación y test sin correr el riesgo de que la tupla simetrica ya se encuentre presente en otro conjunto, asegurando que las tuplas presentes en el conjunto de entrenamiento simetrizado no se encuentren en validación o test, o viceversa. De esta forma, nuestro dataset nos permitiría realizar pruebas con y sin tuplas simétricas con poco esfuerzo. El proceso de eliminación de los simétricos y reflexivos constó en ordenar el dataset alfabéticamente considerando la terna palabra1, palabra2, clasificación, y luego iterar en las tuplas generando el dataset final, donde una tupla fue agregada solo si su simétrico no había sido descubierto aún. Al buscar un simétrico se buscaba por palabras y clasificación, permitiéndonos tener un par de palabras que sean antónimos y sinónimos al mismo tiempo (ej. inhóspito y selvático).

4.3. Análisis de calidad

Luego de armado del dataset procedimos a realizar un análisis de la calidad de los datos obtenidos de nuestras fuentes.

Debido a que para cada pareja de palabras a anotar es necesario buscar su significado en el diccionario de la Real Academia Española y considerar todas sus posibles acepciones se tomo un muestreo de 200 tuplas, de las cuales 100 fueron antónimos y 100 sinónimos. Luego procedimos a realizar la clasificación manual de cada tupla de forma individual.

Una vez clasificadas las tuplas, se calculó la concordancia entre las clasificaciones utilizando la función *Kappa* de Cohen (1960). Al realizar el cálculo con nuestras clasificaciones se obtuvo un puntaje de 0.8999, lo que demuestra una concordancia (y discrepancia) a la hora de deducir que relación corresponde en cada tupla.

Luego se calculó la *accuracy* entre nuestras anotaciones y la clasificación original del dataset, obteniendo un puntaje de 0.9 para ambos. Esto indica que la clasificación obtenida en nuestro dataset sigue la misma línea que la clasificación percibida por una persona al pensar en los significados de las palabras.

Para las tuplas donde no hubo acuerdo entre las anotaciones manuales se realizó un análisis de qué motivo llevó a cada uno a elegir la clasificación y agrupamos las tuplas según el motivo encontrado. Para el caso de las tuplas listadas en la tabla 4.1, nos encontramos con que la misma tupla de palabras podía ser clasificada como antónimo o sinónimos dependiendo los potenciales contextos de las palabras que se estén teniendo en cuenta en el momento de realizar la clasificación. Por ejemplo, selvático puede ser considerado como un lugar hostil a la vida humana por lo cual seria un sinónimo de inhóspito, sin embargo un lugar selvático esta lleno de seres vivos por lo cual es un lugar habitable y por lo tanto antónimo de inhóspito.

Palabra 1	Palabra 2
inhóspito	selvático
enervar	espabilar
lateral	separado
entregarse	ocuparse

Tabla 4.1: Tuplas clasificadas de forma distinta debido a la acepción utilizada por el anotador.

Palabra 1	Palabra 2
idiotizar	embelesarse
perversión	conversión

Tabla 4.2: Tuplas clasificadas de forma distinta debido a la connotación interpretada por el anotador.

Por otro lado, la tuplas listadas en la tabla 4.2, dependiendo de si se tomen los significados con connotación positiva o negativa, pueden ser vistas como antónimos o sinónimos, motivo por el cual generó diferencias entre las clasificaciones. Por ejemplo, si se considera la connotación negativa de conversión entonces perversión y conversión son sinónimos, sin embargo, si consideramos una connotación positiva de conversión entonces conversión y perversión son antónimos.

Un caso particular es la tupla *corrección* y *propiedad*. Estas palabras inicialmente, según nuestro punto de vista, no se encontraban relacionadas, pero al analizar en profundidad sus diferentes significados en el diccionario de la Real Academia Española concluimos en que las mismas son sinónimos, y existen frases como "hablar con corrección" y "hablar con propiedad" donde estas palabras son intercambiables.

4.4. Topología de la sinonimia y antonimia

Se crearon grafos que representaran las relaciones de la antonimia y la sinonimia, y se realizó un análisis de la topología de los grafos obtenidos. Cada grafo contiene las palabras del dataset como vértices, y las aristas se definen de la siguiente manera: para el grafo que representa la antonimia el conjunto de aristas son $(n_1, n_2) \in E_{antónimos}$ si y solo si (n_1, n_2) o (n_2, n_1) son tuplas pertenecientes al dataset como antónimos; las aristas del grafo de la sinonimia se define por $(n_1, n_2) \in E_{sinónimos}$ si y solo si (n_1, n_2) o (n_2, n_1) son tuplas de sinónimos en nuestro dataset. Se utilizó la librería Networkx (Aric Hagberg, 2005) de Python para crear, analizar y graficar los grafos.

En la figura 4.1 se muestra el grafo que representa a la antonimia. Se puede observar en la figura que se encuentra una gran componente conexa (16651 nodos) actuando como núcleo del grafo. Esto quiere decir que a través de la relación

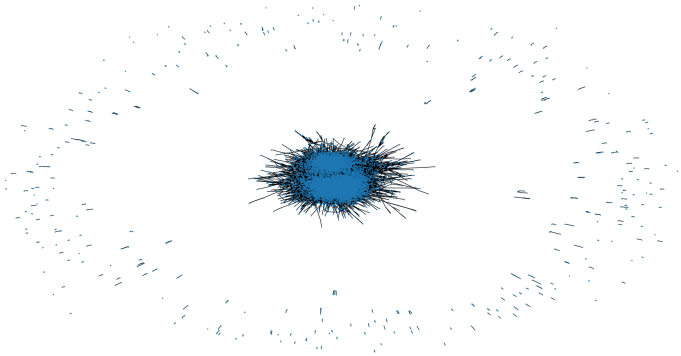

Figura 4.1: Grafo de la relación de **antonimia** generado a partir de dataset. Se puede observar que el grafo cuenta con una gran componente conexa (16651 nodos) la cual esta rodeada por 298 componentes disconexas entre si compuestas de entre 2 y 11 nodos

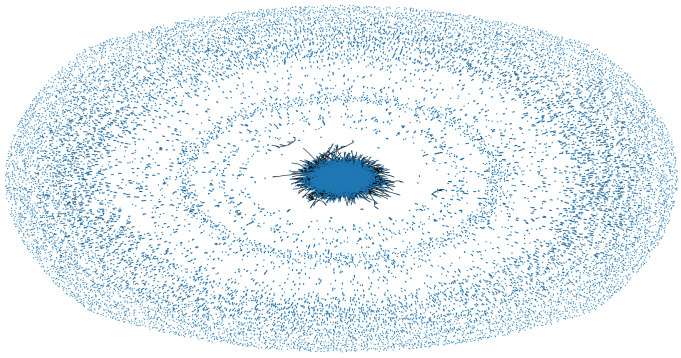

Figura 4.2: Grafo de la relación de **sinonimia** generado a partir del dataset. Al igual que la relación de antonimia, en la sinonimia se observa un núcleo conteniendo 44576 nodos mientras que el conjunto de componentes disconexas esta conformado por 9770 los cuales contienen entre 2 y 20 nodos

de antonimia pueden conectarse 16651 palabras, generándose caminos entre todas ellas. Fuera del núcleo, el grafo contiene 298 componentes conexas, donde cada una contiene entre 2 y 11 nodos. Estas componentes conforman una suerte de nebulosa que en la representación gráfica mostrada rodea el núcleo, sin conectarse entre ellas o con el núcleo.

En la figura 4.2 se presenta el grafo que representa la relación de sinonimia. De forma similar al grafo de antónimos, en el grafo de sinónimos se encuentra una gran componente conexa como núcleo que contiene 44576 vértices y es rodeada por una nebulosa de 9770 componentes conexas de entre 2 y 20 nodos.

A partir de la información que se desprende de los grafos, podemos concluir

que las palabras en el español están muy interconectadas entre si a través de las relaciones léxicas analizadas, debido a que existen caminos dentro de estas relaciones que conectan la mayoría de ellas. Esta alta conectividad dentro del grafo puede ser una razón que explique el por qué de que el corpus de Cardellino no generó un dataset de mayor dimensión que el creado utilizando WordNet como vocabulario inicial. La alta conectividad puede permitir que a partir de un corpus pequeño como WordNet se descubran nuevas palabras y así cubrir la mayoría del grafo.

4.5. Partición del dataset

Luego de tener un conjunto unificado de tuplas con su respectiva clasificación se utilizaron tres estrategias independientes para partirlo. Se consideró la estrategia de selección aleatoria y dos estrategias que no tienen intersección léxica entre los subconjuntos. La partición sin intersección léxica quiere decir que una misma palabra no puede aparecer en conjuntos distintos, es decir, si una palabra aparece en el conjunto de test, no puede aparecer en el de entrenamiento ni en el de validación.

Se decidió realizar particiones sin intersección léxica por dos motivos. El primero es para evaluar el impacto de la "memorización léxica" presentada en Levy et al. (2015) en la detección de antonimia, y el segundo es evaluar el desempeño del clasificador frente a palabras que nunca vio durante el entrenamiento, incrementando la dificultad de la tarea. La "memorización léxica" se da cuando un modelo clasifica a un par como positivos independiente de uno de los términos en la pareja. Por ejemplo, si tenemos las parejas (frío, calor), (frío, cálido), (frío, caluroso) como ejemplos positivos en el conjunto de entrenamiento, el modelo podría aprender a clasificar como ejemplo positivo a cualquier pareja que contenga a la palabra *frío* en el lado izquierdo de la tupla. Este fenómeno es mucho más relevante en la detección de hiperónimos, ya que la relación es propensa a tener hiperonimos masivos. En la antonimia y sinonimia esto no es tan significativo debido a la naturaleza de las relaciones.

Se generaron 2 versiones de las tres particiones con y sin las tuplas simétricas. De esta forma se obtienen un total de 6 particiones diferentes del dataset. En la tabla 4.3 se desglosa la cantidad de elementos de cada partición y conjunto de la versión sin simétricos, para la versión con simétricos la cantidad de tuplas es el doble.

4.5.1. Partición *Random*

Para la partición aleatoria se utilizó la librería *scikit-learn* (Pedregosa et al., 2011). Esta librería posee funciones para la partición aleatoria de datasets, por lo tanto se decidió usar esas funciones para generar las particiones. Este particionamiento fue realizado de forma estratificada para preservar los porcentajes del

		Ant	**Syn**	**Total**
Rand	Train	47720	244378	292098
	Val	3592	18394	21986
	Test	12828	65693	78521
Lex Shwartz	Train	28571	132772	161343
	Val	3684	18803	22487
	Test	7498	37503	45001
Lex Graph	Train	27124	77200	104324
	Val	6647	18919	25566
	Test	13357	38017	51374

Tabla 4.3: Cantidades de los conjuntos en su versión sin simétricos

dataset original (17 % ant, 83 % syn), es decir la proporción de antónimos y sinónimos presentes en cada uno. Para el conjunto de entrenamiento se asigno un 70 % de las tuplas del dataset, para validación un 10 % y para test un 20 %.

4.5.2. Partición Léxica: Shwartz

Esta partición del dataset está basado por la implementación realizada en Shwartz et al. (2016). Nuestra implementación constó en iterar sobre el dataset y cada tupla es asignada a entrenamiento, validación o test tratando de preservar los porcentajes 70 %, 10 % y 20 % respectivamente. Es decir, cada 7 tuplas agregadas en el conjunto de entrenamiento se agregó una en validación y dos en test. Si una tupla no podía ser asignada a uno de los tres conjuntos debido a que no cumplía la condición de partición sin intersección léxica, esta es descartada. En la tabla 4.3 se pueden ver las cantidades para la partición sin simétricos.

A continuación se presenta una pseudo-código para la implementación realizada:

Algorithm 1 Separación léxica Shwartz

$train \leftarrow EmptySet$
$val \leftarrow EmptySet$
$test \leftarrow EmptySet$
for (w_1, w_2) *in Dataset* **do**
  $can\_be\_in\_train \leftarrow w_1 \notin test \wedge w_2 \notin test \wedge w_1 \notin val \wedge w_1 \notin val$
  $can\_be\_in\_test \leftarrow w_1 \notin train \wedge w_2 \notin train \wedge w_1 \notin val \wedge w_1 \notin val$
  $can\_be\_in\_val \leftarrow w_1 \notin train \wedge w_2 \notin train \wedge w_1 \notin test \wedge w_1 \notin test$
  if $\neg can\_be\_in\_train \wedge \neg can\_be\_in\_test \wedge \neg can\_be\_in\_val$ **then**
    Descartar (w_1, w_2)
  else
    Asignar manteniendo proporciones.
  end if
end for

4.5.3. Partición Léxica: Grafo

Para la creación de esta partición, nos enfocamos en el grafo de la relación de antonimia. Creamos un algoritmo basado en la estructura del grafo de la figura 4.1 que intentara minimizar la cantidad de tuplas descartadas al realizar la partición sin intersección léxica.

El primer paso fue separar el grafo en componentes conexas. Dado que cada palabra está representada por un nodo, al utilizar distintas componentes conexas nos aseguramos la partición léxica entre dichos conjuntos, ya que distintas componentes conexas no comparten nodos. Como se mencionó anteriormente en la sección 4.4, el grafo cuenta con una gran componente conexa (núcleo) y muchas componentes conexas pequeñas. Estas pequeñas componentes no alcanzan a ser ni 10 % de los antónimos presentes en el dataset y decidimos destinarlas a formar parte del conjunto de test.

Posteriormente se probaron varias soluciones para separar el resto del grafo de forma que se perdiera la menor cantidad de tuplas del dataset. En esta sección procederemos a explicar la versión final del algoritmo, en el apéndice A se encuentran explicados los diferentes algoritmos que se desarrollaron y derivaron en la versión final.

La solución final del algoritmo de particionamiento toma el núcleo del grafo y a partir de él genera primero los conjuntos de validación y test, y luego toma las tuplas restantes como conjunto de entrenamiento. Éste algoritmo consta de aplicar un *Breadth-first search* (BFS) sobre el núcleo.

Un BFS es un recorrida en amplitud que se realiza sobre los nodos del grafo. Para esto, se toma un nodo como raíz y se procede a explorar todos los demás

nodos del grafo, comenzando en los adyacentes al nodo raíz, luego a los adyacentes de los adyacentes y así sucesivamente. En la figura 4.3 se puede observar el orden en el que se recorrerían los nodos en un grafo.

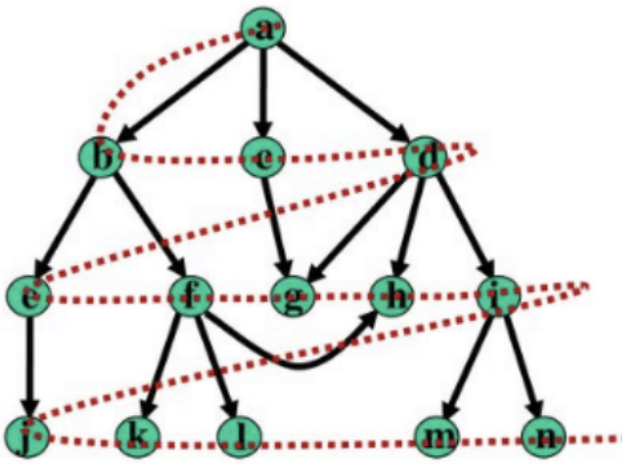

Figura 4.3: Recorrida BFS de un grafo.[4]

Nuestro algoritmo toma el nodo más conectado como raíz y realiza la búsqueda comenzando desde éste, añadiendo aristas y nodos a la lista de visitados hasta completar la cantidad de antónimos necesaria en test. La lista de nodos (y aristas) es desconectada del resto del grafo y asignada al conjunto de test, pudiendo de esta manera repetir el proceso para armar validación.

Una vez armado validación y extraídos esos nodos del grafo, todo el grafo restante es asignado al conjunto de entrenamiento. De esta forma logramos que test y validación tengan el 20 % y 10 % de la cantidad de tuplas antónimos originales respectivamente, en cambio el conjunto de entrenamiento tendrá el 70 % menos la cantidad de tuplas descartadas.

Finalmente se agregan las tuplas de sinónimos a los tres conjuntos, respetando que ambas palabras de la tupla se encuentren presentes en el conjunto donde se van a agregar. Se agregan tuplas de forma tal que los antónimos representen un 25 % del total de cada conjunto, es decir una proporción de 1:3.

[4]Imagen extraída de `https://www.junhaow.com/studynotes/cs61b/cs61b%20p9`

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 5

Experimentos

En este capítulo presentamos los modelos creados y los experimentos realizados. En todos los experimentos se utiliza el dataset previamente creado, vectorizando las palabras utilizando una versión preentrenada de FastText (Joulin et al., 2016) con vectores de 300 dimensiones. El entrenamiento de esta versión de FastText fue realizada con la versión en español de Wikipedia. Para realizar los experimentos se utilizan 3 modelos, una linea base llamada concat, que consiste en una red *feed-forward* que toma como entrada la concatenación de los vectores de las palabras, una red siamesa y la red Parasiamesa. Para cada modelo y dataset se realizó *Random Search* para la búsqueda de hiperparámetros, habiéndose ejecutado más de 40 experimentos.

5.1. Modelos

Todos los modelos utilizados para los experimentos utilizan una red neuronal de tipo *feed-fordward* completamente conectada. La configuración de la red base es parte del conjunto de hiperparámetros de los modelos usados. Algunos de estos parámetros son, la cantidad y el tamaño de las capas y la función de activación entre cada capa. Para la implementación de los modelos se utilizó la librería PyTorch (Paszke et al., 2019) de Python.

5.1.1. Base line: Concat

El modelo utilizado como linea base es una red neuronal de tipo *feed-fordward* cuya entrada es la concatenación de los vectores de las palabras de la tupla, es decir, dado el par (x, y), la entrada de la red es el vector $(x_1, .., x_{300}, y_1, .., y_{300})$, y su salida consta de una única neurona con función de activación sigmoid, por lo que los valores retornados por la red se encuentran entre 0 y 1.

En este modelo los antónimos fueron tomados como clase positiva, y los sinónimos como clase negativa. Como la salida del mismo está dada por valores continuos

entre 0 y 1, para definir a qué clase corresponde la clasificación, se aplica un redondeo matemático a la salida, mapeando los valores a 0 (clase sinónimo) si es menor a 0,5 y a 1 si es mayor o igual a 0,5 (clase antónimo). La figura 5.1 presenta un diagrama de la red utilizada.

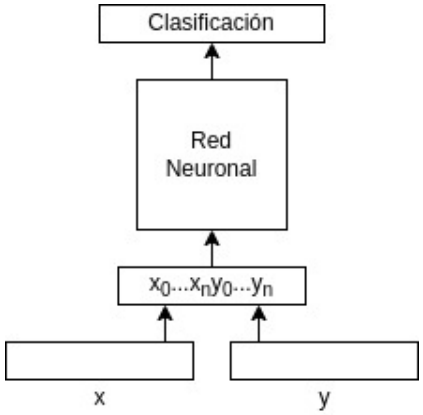

Figura 5.1: Diagrama de la red neuronal correspondiente al modelo concat.

5.1.2. Siamesa

Los modelos presentados en esta sección siguen la estructura de las redes neuronales siamesas presentadas en la sección 2.1.1.3. La red neuronal interna utilizada es un red *feed-forward* completamente conectada. Tiene como entrada dos vectores, uno por cada palabra de la tupla. Estos vectores son aplicados a la red neuronal base y la distancia euclidea al cuadrado de las salidas es el resultado final de la red siamesa. La distancia euclídea al cuadrado está dada por la siguiente formula:

$$d(x, y) = (x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_n - y_n)^2$$

Para entrenar y obtener la clasificación a partir de la distancia provista por la red se utilizan tres márgenes, donde dos de estos márgenes son los utilizados en la función de pérdida y el tercero, llamado "acceptance", que cumple con $margen\_neg >= acceptance >= margin\_pos$ que se utiliza para clasificar. Consideramos los siguientes tres criterios de clasificacion:

- Strict: utiliza el margen positivo como margen de aceptación.
- Lenient: utiliza el margen negativo como margen de aceptación.
- Acceptance: utiliza el valor *acceptance* como margen de aceptación.

Una vez definido el margen de aceptación (según el criterio), si la distancia entre los vectores transformados es menor o igual a dicho valor se clasifica al ejemplo como perteneciente a la clase positiva, y de lo contrario se clasifica en la clase negativa. Se utilizaron dos variantes de la red siamesa, siamesa ANT y siamesa SYN que explicamos a continuación.

5.1.2.1. Siamesa ANT

En esta variante de la red siamesa se minimiza la distancia entre los vectores de las palabras que son antónimos y se aumenta para los sinónimos. Por lo tanto, son clasificadas como antónimos (clase positiva) las tuplas de palabras cuyos vectores transformados se encuentren a menor distancia que el margen de aceptación considerado por el criterio, y son clasificados como sinónimos si lo superan.

5.1.2.2. Siamesa SYN

Esta variante, en contraparte, intenta disminuir la distancia entre los vectores transformados de los sinónimos y aumentarla entre los vectores de los antónimos. Es decir, si la distancia es menor al margen de aceptación considerado por el criterio se clasifica como sinónimo y si lo supera como antónimo. La denominación de los modelos se debe a que esta variante de la red siamesa puede considerarse como una red siamesa que detecta sinónimos.

5.1.3. Parasiamesa

Con intenciones de probar la red Parasiamesa (3.2.1) aplicado en el idioma español, se implementó una red neuronal de tipo parasiamesa siguiendo la estructura presentada. Como se mencionó, al igual que la red siamesa, la red parasiamesa tiene una red neuronal interna que se aplica a las entradas de la misma. En nuestro modelo se utilizó la misma red *feed-fordward* base utilizada en la red siamesa. Además las funciones de pérdida y de distancia utilizadas fueron las mismas que las de la red siamesa.

Con este modelo se buscó disminuir la distancia entre los vectores transformados de aquellas tuplas relacionadas por la antonimia, y aumentarla entre los trasformados de las tuplas de sinónimos. Para clasificar a las tuplas se utilizó el mismo procedimiento que el utilizado en el modelo Siamesa.

Debido a su estructura no simétrica donde la red interna se aplica dos veces en una de sus entradas, implementamos dos variantes: parasiamesa *left*, que aplica dos veces la transformación de la red base sobre el vector izquierdo de la tupla, y la parasiamesa *right* que hace lo mismo sobre el lado derecho.

5.1.3.1. Parasiamesa pre-entrenada

Además de las variantes *left* y *right* se implementó la posibilidad de realizar un pre-entrenamiento de la red interna. El mismo consta de entrenar la red interna como una red Siamesa SYN. Posteriormente los pesos finales de esta red son utilizados como los pesos iniciales de la red interna de la red parasiamesa. El pre-entrenamiento puede realizarse tanto en la versión *right* como *left*.

5.2. Random Search

Para cada variante del dataset y modelo se realizaron búsquedas de hiperparámetros independientes mediante *random search*. Esta técnica consiste en realizar un muestreo aleatorio del espacio de hiperparámetros. Esto se realizó utilizando los datasets con y sin simétricos. Los *random searches* consistieron en probar 200 combinaciones distintas de hiperparámetros, y cada entrenamiento fue detenido utilizando *early stopping*. *Early stopping* consiste en detener el entrenamiento de un modelo si este no mejora su desempeño en el conjunto de validación luego de una determinada cantidad de iteraciones (*epochs*), donde a ésta se la conoce como paciencia. En este *random search* se utilizó *early stopping* con una paciencia de 2 *epochs* para el modelo *concat* y 3 *epochs* para los demás. En todos los casos se utilizó *Adam* (Kingma and Ba, 2014) para entrenar los diferentes modelos y la estrategia utilizada fue "acceptance". Para realizar las ejecuciones de los *random search* se utilizó la librería RayTune (Liaw et al., 2018).

Inicialmente se utilizó un espacio de búsqueda detallado en el apéndice B. A partir de observaciones realizadas en las primeras ejecuciones efectuadas con este espacio de búsqueda se realizaron modificaciones y se obtuvo el siguiente espacio de búsqueda:

- Cantidad de capas de la red neuronal en el rango de $[1, 4] \in N$.
- Tamaño de las capas $[100, 700] \in N$. Un valor para cada capa es seleccionado independientemente.
- Tasa de aprendizaje (Learning rate): uno dentro de 1e-2, 1e-3, 1e-4
- Función de activación entre capas: una de [ReLU, Tanh]
- Tamaño de lote (Batch size), uno de: 256, 512, 1024, 2048, 4096
- Márgenes, utilizados para el *constrasting loss* en los modelos siamesa y parasiamesa, y el valor *acceptance* utilizado para la clasificación. Estos valores se toman del rango $[0{,}01, 0{,}02, ..., 7]$.

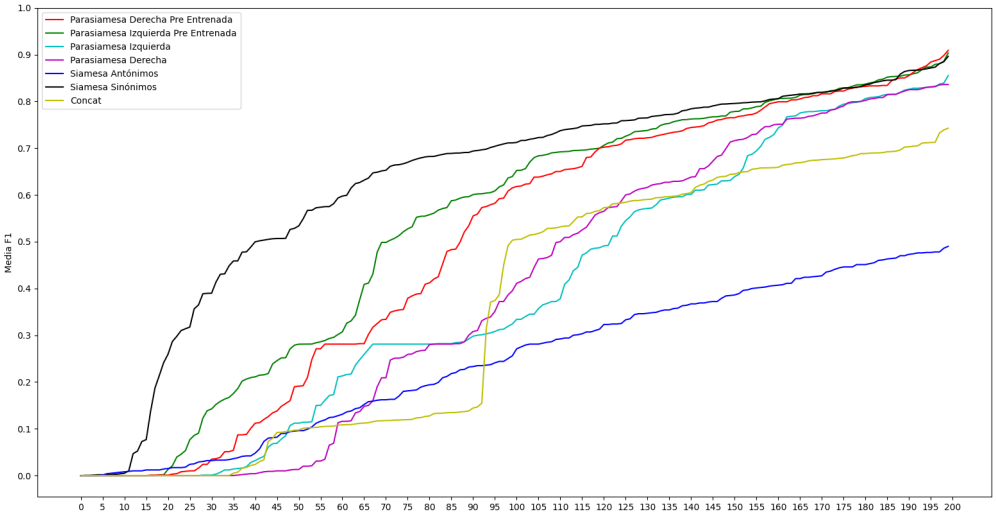

Figura 5.2: Medida F1 en el conjunto de validación, de diferentes ejecuciones del *random search* en la partición *random*

5.2.1. Análisis de las búsquedas

En la figura 5.2 se muestra la mejor medida F1 obtenida en el conjunto de validación para el *random search* en los distintos modelos. Los datos de la gráfica se encuentran ordenados por medida F1 y no por número de ejecución, de esta forma el n^{esimo} elemento es siempre la n^{esima} peor configuración del modelo. Como se puede ver en la gráfica, el modelo de Siamesa en versión sinónimos se eleva rápidamente, lo cual indica que este modelo es menos sensible a los parámetros debido a que para la mayoría de las configuraciones se encuentra por encima de los demás modelos.

Se puede ver en la figura 5.2 que las versiones pre-entrenadas de la red Parasiamesa siempre se encuentran por encima de las versiones no pre-entrenadas. Esta observación nos lleva a pensar que el modelo pre-entrenado tiende a aprender mejor la relación de antonimia que las versiones no pre-entrenadas, independientemente de los parámetros.

De todos los modelos, el Siamesa ANT es el que se encuentra más alejado de todos los demás, estando los mejores resultados de este modelo un 30 % por debajo a los resultados obtenidos por los demás.

En el caso de la línea base *concat*, puede observarse que aproximadamente la mitad de las configuraciones dan resultados muy bajos (menores a 0.20). La otra mitad da resultados que rondan entre 0.5 y 0.6 aproximadamente.

La red Siamesa de sinónimos se encuentra por encima de los demás modelos para

la mayoría de las configuraciones. Sin embargo este modelo reinterpreta el problema de la detección de antonimia. Con lo cual podría no funcionar adecuadamente en un dataset con tuplas de palabras no relacionadas.

Los valores máximos de F1 fueron obtenidos por las redes Parasiamesas pre-entrenadas. Estos modelos fueron los que obtuvieron un promedio de F1 mayor en todas las configuraciones luego de la siamesa sinónimos.

5.2.2. Mejores configuraciones

En las tablas 5.3, 5.2 y 5.1 se presentan las mejores configuraciones encontradas para cada conjunto de datos en su versión sin simétricos, para cada uno de los modelos.

Modelos	Capas	Lr	Funcion	T batches	Margenes
Concat	[583, 624, 381]	0.001	ReLU	64	-
Siam ANT	[633, 217]	0.001	ReLU	1024	[1.51, 5.85, 6.58]
Siam SYN	[571, 382, 122]	0.001	ReLU	256	[1.26, 2.79, 3.81]
PSiam R	[654]	0.001	ReLU	248	[2.95, 4.41, 5.15]
PSiam L	[601]	0.0001	ReLU	256	[1.53, 3.33, 3.96]
PSiam Pre R	[598, 477, 619]	0.001	ReLU	4096	[2.72, 4.69, 6.69]
PSiam Pre L	[636, 237, 550]	0.001	ReLU	1024	[1.99, 3.25, 4.2]

Tabla 5.1: Mejores combinaciones de hiperparámetros encontradas al utilizar la partición *random* sin simétricos

Modelos	Capas	Lr	Funcion	T batches	Margenes
Concat	[585, 664, 598]	0.0001	ReLU	128	-
Siam ANT	[644, 349]	0.001	ReLU	2048	[0.45, 3.58, 3.71]
Siam SYN	[562, 366, 263]	0.001	ReLU	256	[2.17, 2.21, 4.33]
PSiam R	[420, 619]	0.001	ReLU	1024	[0.43, 4.83, 6.95]
PSiam L	[624]	0.001	ReLU	2048	[1.3, 3.17, 3.52]
PSiam Pre R	[688, 529, 354]	0.001	ReLU	256	[1.14, 3.52, 3.81]
PSiam Pre L	[688, 529, 354]	0.001	ReLU	512	[1.14, 3.52, 3.81]

Tabla 5.2: Mejores combinaciones de hiperparámetros encontradas al utilizar la partición con separación léxica basada en Shwartz sin simétricos.

Se puede ver que en todas las tablas (exceptuando la siamesa antónimos para el dataset del grafo) la función de activación utilizada en todos los casos es ReLU, y en la mayoría de los casos el *learning rate* es de 0.001. En cuanto a los tamaños y

Modelos	Capas	Lr	Funcion	T batches	Margenes
Concat	[542, 332, 670]	0.0001	ReLU	64	-
Siam ANT	[540, 347, 684]	0.001	TanH	256	[5.43, 5.94, 6.47]
Siam SYN	[562, 366, 263]	0.001	ReLU	256	[2.17, 2.21, 4.33]
PSiam R	[659, 571]	0.001	ReLU	256	[1.63, 4.88, 5.0]
PSiam L	[125, 481]	0.001	ReLU	256	[0.34, 1.5, 1.79]
PSiam Pre R	[376, 408, 650]	0.001	ReLU	256	[0.89, 4.49, 5.76]
PSiam Pre L	[420, 619]	0.001	ReLU	512	[0.43, 4.83, 6.95]

Tabla 5.3: Mejores combinaciones de hiperparámetros encontradas al utilizar la partición generada con el algoritmo basado en el grafo sin simétricos.

cantidades de capas, hay mayor variación entre los modelos, desde instancias con una sola capa oculta de unos 600 elementos a instancias con varias capas ocultas de diversos tamaños.

En algunos casos se puede ver que la mejor configuración para distintos modelos coincide. Por ejemplo, para el conjunto basado en Shwartz, las redes parasiamesas pre-entrenadas derecha e izquierda coinciden en todos los parámetros excepto el tamaño del *batch*. Lo mismo sucede entre las redes parasiamesa derecha en este conjunto con la parasiamesa pre-entrenada izquierda en el conjunto basado en el grafo. El caso más particular se da en la siamesa sinónimos, también entre estos conjuntos, donde las mejores configuraciones utilizan los mismos hiperparámetros en su totalidad.

Se observa que las configuraciones similares se dan únicamente entre los conjuntos sin intersección léxica entre entrenamiento y validación. Esto nos lleva a pensar que la separación léxica favorece determinadas configuraciones, motivo por el cual no hay configuraciones similares a las del conjunto *random*.

5.2.2.1. Análisis del entrenamiento

En esta sección nos enfocaremos en estudiar la evolución de los diferentes modelos durante el entrenamiento. Con este objetivo se presentan las figuras 5.3, 5.4 y 5.5. En estas gráficas se puede observar, para cada iteración del entrenamiento, la medida F1 en el conjunto de entrenamiento (con linea entera) y la medida F1 en el conjunto de validación (con linea entrecortada).

En la figura 5.3 se muestra la evolución del entrenamiento de todos los modelos sobre la partición *random* sin elementos simétricos. En esta gráfica se puede apreciar que, en su mayoría, todos los modelos poseen un comportamiento similar. Los mismos empiezan con un puntaje relativamente bajo, y luego de 3 o 4 iteraciones aumentan bruscamente. Luego, en sucesivas iteraciones, continúan teniendo altos y bajos, pero estos no son tan pronunciados como los vistos al principio.

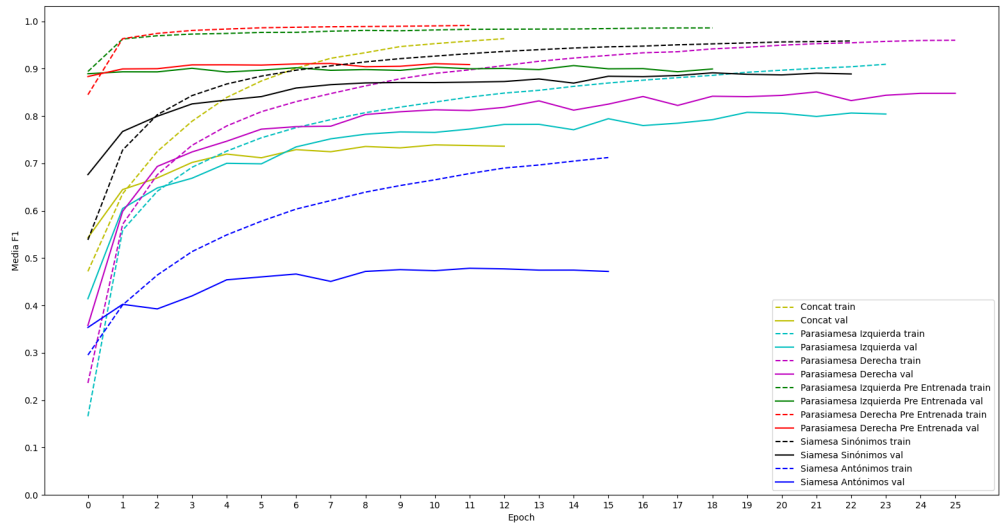

Figura 5.3: Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición *random*

En la figura 5.4 se muestra la evolución del entrenamiento de todos los modelos sobre la partición de grafo. En esta gráfica se puede apreciar que el comportamiento es mas errático que en la figura 5.3. Se puede ver como casi todos los modelos (excepto por las parasiamesas pre-entrenadas) tienen considerables altos y bajos cada pocas iteraciones.

En la figura 5.5 se muestra la evolución del entrenamiento de todos los modelos sobre la partición pasada en Shwartz. Se puede observar en la figura que ningún modelo logra superar una medida F1 de 0.55 aproximadamente en el conjunto de validación, aunque sí aumente en el conjunto de entrenamiento. Esto nos lleva a pensar que en esta partición el conjunto de entrenamiento no posee la suficiente información para que los modelos aprendan correctamente la relación de antonimia.

En el apéndice C se encuentran las gráficas correspondientes a los modelos entrenados con los conjuntos simetrizados. Estas gráficas no fueron incluidas en esta sección debido a que no aportaron información relevante, ya que los modelos se comportaron de forma similar.

5.3. Análisis de resultados

En total se realizaron 36 *random searches*, con los cuales se entrenaron un total de 7200 modelos. Una vez obtenidos los mejores modelos para cada tipo de particionamiento y conjunto se procedió a evaluar sus desempeños, utilizando los

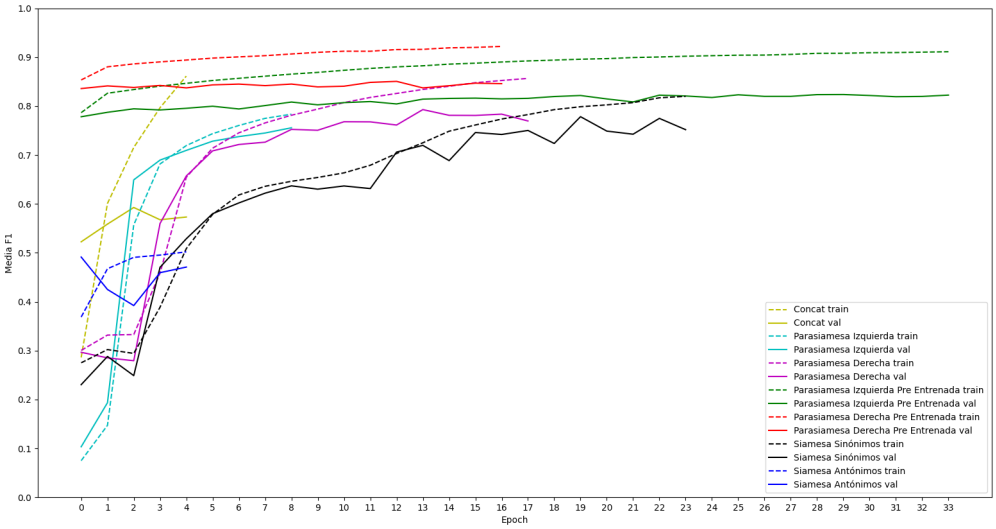

Figura 5.4: Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición basada en grafo

conjuntos de *test* simetrizados correspondientes a cada tipo de particionamiento. En la tabla 5.4 se muestran los resultados para las ejecuciones realizadas con los datasets sin simétricos, y en la tabla 5.5 se encuentran los resultados de las ejecuciones que utilizan los conjuntos de entrenamiento y validación simetrizados. Ambas tablas contienen las medidas precisión (P), *recall* (R) y F1 para el mejor modelo encontrado en el *random search*, evaluados en el conjunto de *test* con simétricos en todos los casos. En estas tablas, los nombres de los modelos se encuentran abreviados, siendo *Siam* siamesa, *PSiam* parasiamesa, *R* right y *L* left, *PSiam Pre* parasiamesa preentrenada.

Models	Random			Lex Shwartz			Lex Graph		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Concat	0.69	0.66	0.68	**0.59**	0.41	0.48	0.59	0.34	0.43
Siam ANT	0.41	0.58	0.48	0.30	0.45	0.36	0.26	**1.00**	0.41
Siam SYN	**0.93**	0.84	0.88	0.54	0.57	0.56	0.61	0.50	**0.55**
PSiam R	0.85	0.83	0.84	0.57	0.51	0.53	0.57	0.40	0.47
PSiam L	0.79	0.84	0.81	0.48	**0.62**	0.54	**0.62**	0.37	0.46
PSiam Pre R	**0.93**	0.87	**0.90**	0.52	**0.62**	0.56	0.59	0.47	0.52
PSiam Pre L	0.91	**0.88**	0.89	0.58	0.58	**0.58**	0.58	0.48	0.53

Tabla 5.4: Métricas obtenidas al evaluar en conjuntos de *test* simetrizado los modelos entrenados con conjuntos sin simétricos ni reflexivos

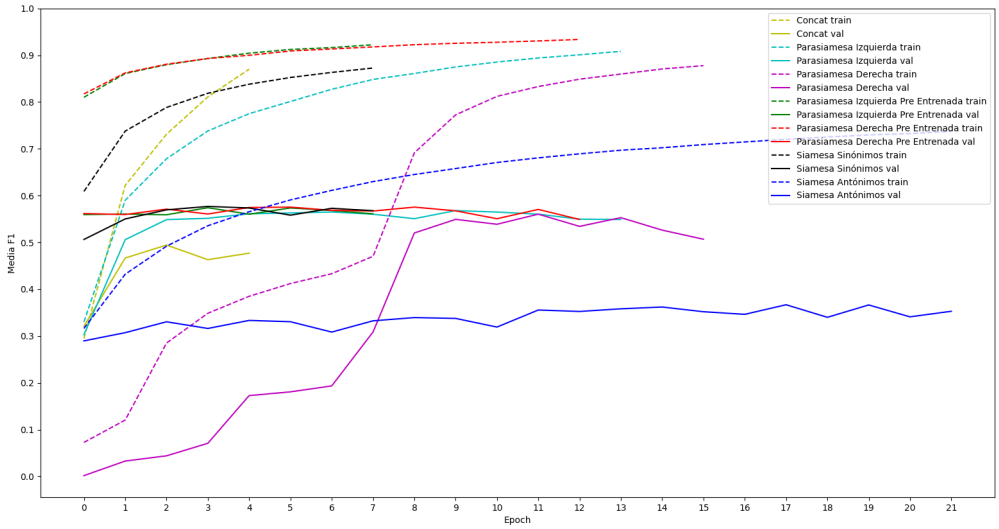

Figura 5.5: Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición basada en *Shwartz*

Models	Random			Lex Shwartz			Lex Graph		
	P	**R**	**F1**	**P**	**R**	**F1**	**P**	**R**	**F1**
Concat	0.80	0.76	0.78	**0.58**	0.50	0.53	**0.64**	0.33	0.44
Siam ANT	0.41	0.58	0.48	0.30	0.45	0.36	0.26	**1.00**	0.41
Siam SYN	**0.93**	0.84	0.88	0.54	0.57	0.56	0.61	0.50	**0.55**
PSiam R	0.88	0.85	0.87	0.55	0.57	0.56	0.61	0.45	0.52
PSiam L	0.89	0.87	0.88	0.47	0.65	0.55	0.58	0.40	0.47
PSiam Pre R	0.92	0.88	**0.90**	0.46	**0.66**	0.55	0.61	0.50	**0.55**
PSiam Pre L	0.88	**0.89**	0.89	0.56	0.58	**0.57**	0.60	0.49	0.54

Tabla 5.5: Métricas obtenidas al evaluar en conjuntos de *test* simetrizado los modelos entrenados con conjuntos de entrenamiento y validación también simetrizados.

Como puede observarse en la tabla 5.4, el mejor modelo para los entrenamientos con el dataset sin simetrizar, en términos de F1, en las particiones *random* y basada en Shwartz, es la parasiamesa preentrenada, mientras que en la partición basada en el grafo es la siamesa sinónimos. Las variantes *left* y *right* se encuentran muy cercanas en cuanto a medida F1, teniendo una diferencia máxima de 0.03 en todas las particiones, lo que indica que el lado de doble aplicación de la red base no afecta significativamente los resultados. Si se compara la versión preentrenada con la versión sin preentrenar, se encuentra una diferencia mayor, ya que la versión preentrenada aumenta la medida F1 entre un 0.03 y un 0.08, generando una buena mejora considerando las pequeñas modificaciones necesarias para preentrenar el modelo.

Observando la tabla 5.5, puede notarse que casi todos los modelos fueron beneficiados al ser entrenados con los conjuntos simetrizados. El único modelo que no mejoró notoriamente fue la versión preentrenada de la parasiamesa. Debido a la naturaleza simétrica que poseen las redes siamesas, no se realizaron *random search* de este modelo utilizando los conjuntos simetrizados, por lo que los resultados para las redes siamesas expresados en 5.5 son los mismos que en 5.4 y fueron incluidos para facilitar las comparaciones. El mejor modelo en términos de medida F1, continúa siendo la parasiamesa para las particiones basadas en Shwartz y el grafo. Para la partición *random*, la parasiamesa preentrenada derecha y la siamesa sinónimos fueron los modelos con mejor desempeño. Para estos conjuntos, la distancia entre las F1 de la versión preentrenada y sin preentrenar de la parasiamesa fue más acotada, variando entre 0.01 y 0.03, exceptuando la versión izquierda en la partición léxica basada en el grafo, donde la diferencia fue de 0.07.

Si comparamos los resultados del mismo modelo en ambas tablas, podemos ver que tanto *concat* como la parasiamesa no preentrenada mejoran su comportamiento. Esto se corresponde con que ninguno de ambos modelos es simétrico, por lo que son dependientes del orden de las palabras dentro de la tupla. En el caso de la parasiamesa, aplica dos veces la red base solo sobre uno de los elementos de la tupla. Si tenemos la tupla simétrica, el modelo generará un nuevo par de vectores transformados para el mismo par de palabras. En el caso del modelo concat la explicación es similar, como utiliza la concatenación de los vectores asociados a las palabras de la tupla como entrada, si tenemos la tupla simétrica obtendremos un vector de entrada diferente.

Por otro lado, la parasiamesa preentrenada parece no mejorar al utilizar conjuntos con simétricos. Creemos que esto puede ser explicado por el preentrenamiento del modelo. Como este está siendo preentrenado con la siamesa sinónimos, y este modelo es simétrico por naturaleza, creemos que es capaz de aprender las tuplas simétricas sin tenerlas dentro del conjunto de entrenamiento. Por lo tanto, si se necesita utilizar un dataset que no puede ser simetrizado, preentrenar la red parasiamesa nos garantizaría un resultado de similar desempeño.

Se puede ver en ambas tablas que el desempeño de los modelos es notoriamente inferior en los conjuntos sin intersección léxica. Consideramos que esto se debe a la dificultad que representa para los modelos inferir sobre una palabra que no ha visto anteriormente.

Esta página ha sido intencionalmente dejada en blanco.

Capítulo 6

Conclusión y trabajos futuros

En este trabajo se abordó el problema de detección de antónimos y sinónimos en el idioma español. Dado que no existen antecedentes o recursos para este idioma, se creó un dataset anotado que contiene tuplas de antónimos y sinónimos a partir de recursos web. Se realizó una anotación manual de un muestreo del dataset generado para evaluar su calidad.

Se generaron tres versiones del dataset según su separación en entrenamiento, validación y *test*; una de ellas de forma aleatoria y las otras dos sin que hayan palabras en común entre las distintas particiones, es decir, sin intersección léxica. Para las particiones sin intersección léxica se utilizó: (1) una implementación propia del algoritmo de Shwartz et al. (2016), y (2) un nuevo algoritmo diseñado a partir de la visualización del grafo de la relación de antonimia observando que tiene una gran componente conexa de mas de 10000 nodos y las restantes son componentes mucho más pequeñas con menos de 10 nodos. Por otro lado para cada versión del dataset se da una variante con y sin tuplas simétricas.

Se llevaron a cabo experimentos con el dataset previamente construido utilizando la red parasiamesa (Etcheverry and Wonsever, 2019), realizándose para esto una implementación completa del modelo. Con el objetivo de comparar el desempeño de la red parasiamesa se implementó una red neuronal *feed-forward* que utiliza como entrada la contención de la representación vectorial de las palabras. Además, se implemento una red siamesa que utilizamos tanto para sinónimos como antónimos.

Con el fin de encontrar una buena configuración de hiperparámetros para cada modelo y variante del dataset, se realizaron búsquedas aleatorias independientes de 200 combinaciones distintas de hiperparámetros (7800 en total), realizando un análisis comparativo entre los entrenamientos de los distintos modelos. Luego, se evaluaron los modelos sobre los conjuntos de *test* correspondientes y se realizó un análisis de los resultados obtenidos. Este análisis llevó a concluir que el modelo de red parasiamesa preentrenada es el más eficaz de los modelos implementados para distinguir los sinónimos de los antónimos en la mayoría de los conjuntos, obteniendo una medida F1 de 0.9 para la partición *random* con y sin simétricos,

0.58 en la partición *Lex Shwartz* sin simétricos y 0.57 en la versión con simétricos, y para la partición *Lex Graph* obtuvo 0.53 y 0.55 en la versión sin y con tuplas simétricas respectivamente. Dejando en evidencia que las variantes sin intersección léxica presentan una tarea más desafiante para los modelos utilizado.

Como trabajos futuros se plantean: (1) evaluar con el dataset creado otros modelos existentes en la literatura, que han sido utilizados originalmente para el inglés, y (2) extender la tarea de ASD con la inclusión de palabras que no son antónimas ni sinónimas, llevando así la tarea de clasificación binaria a una clasificación con tres clases (antónimos, sinónimos y no-relacionado) e invalidando la posibilidad de abordar la tarea como la clasificación de (sinónimo, no-sinónimo).

Apéndice A

Partición léxica: Grafo

En este apéndice nos enfocamos detallar el proceso de armado del algoritmo de particionamiento basado en el grafo de la relación de antonimia, explicando los pasos intermedios realizados para llegar a la versión final. Con el algoritmo se buscó seleccionar qué tuplas del conjunto de antónimos se agregaba en entrenamiento, validación y test, y posteriormente se añadieron los sinónimos manteniendo partición léxica.

Como se explicó en la sección 4.5.3, el primer paso fue asignar las componentes conexas de la periferia al conjunto de test. Luego se procedió a completar este conjunto y armar entrenamiento y validación. Dado que el conjunto de entrenamiento es el más grande, se decidió armar primero este conjunto, y luego validación y test.

Si se utiliza un algoritmo que simplemente tome aristas del núcleo y las agregue en el conjunto de entrenamiento, el costo de mantener la partición léxica generaba una pérdida de tuplas muy grande, por lo que se utilizaron conceptos de teoría de grafos que nos permitiera construir un algoritmo más inteligente.

En un grafo, un punto de articulación, como muestra la figura A.1, es aquel vértice tal que si se elimina se incrementa la cantidad de componentes conexas. Sabiendo esto se intentó eliminar todas las aristas adyacentes a los puntos de articulación menos una, de forma de no aislar el vértice, ya que de hacerlo la palabra no quedaría en ninguna tupla y la perderíamos de nuestro vocabulario. Se realizó este procedimiento hasta obtener componentes conexas lo suficientemente pequeñas, de manera que al juntarlas su cardinalidad de aristas se asemeje a la cantidad de tuplas necesarias en el conjunto de entrenamiento. La pérdida de tuplas generada con este enfoque seguía siendo demasiado grande, generando conjuntos de validación y test muy pobres.

Otro concepto útil para este algoritmo fueron las aristas de corte. Como se puede observar en la figura A.2, una arista de un grafo es arista de corte si al eliminarse se incrementa la cantidad de componentes conexas del mismo. De igual manera que

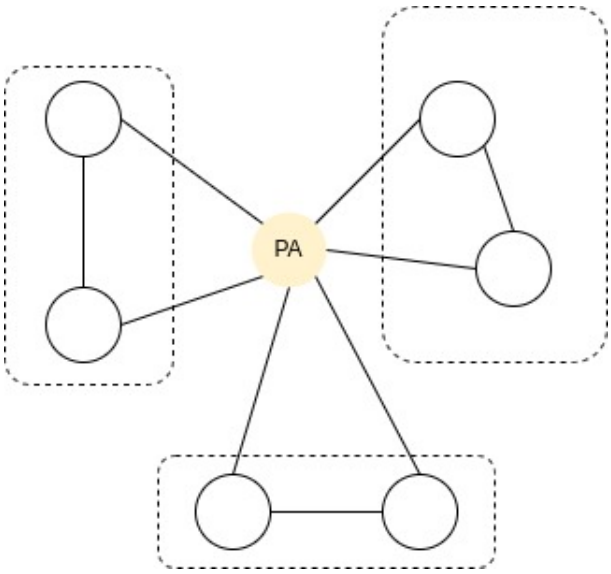

Figura A.1: Ejemplo de articulación de un grafo y las componentes conexas generadas si el mismo es eliminado.

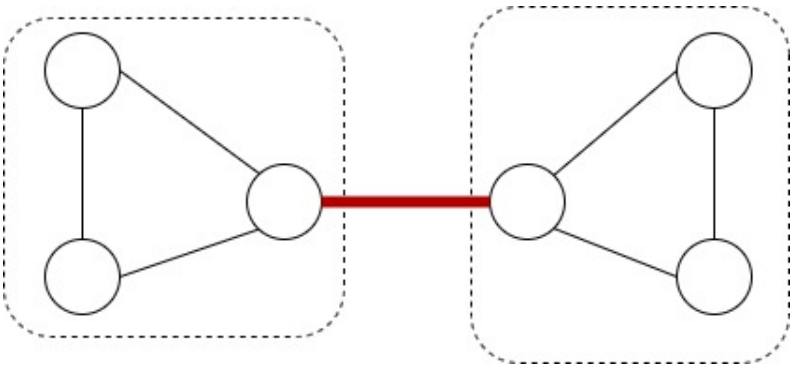

Figura A.2: Ejemplo de arista de corte de un grafo y las componentes conexas generadas si la misma es eliminada.

los puntos de articulación, estas aristas nos permiten desconectar el grafo, ya que eliminando una única arista generamos una componente conexa nueva. Eliminando todas las aristas de corte de nuestro núcleo no se llegó a generar componentes lo suficientemente pequeñas para armar el conjunto de entrenamiento, lo que nos llevo a generar un algoritmo que juntara los dos enfoques, utilizando primero aristas de corte y posteriormente puntos de articulación. Sin embargo, no se logró mejorar los resultados obtenidos al utilizar solo puntos de articulación.

Dado que al eliminar aristas generábamos nuevas componentes conexas pequeñas, se intento invertir el orden en los que se generaban los conjuntos, realizando

primero test y validación, y por último el conjunto de entrenamiento. Al realizar este procedimiento, debido a la pérdida de tuplas, el conjunto de entrenamiento resultaba muy pequeño.

Siguiendo la línea de asegurar la cantidad de tuplas de validación y test, se implementó una última variante del algoritmo que generaba estos conjuntos, y asignaba las tuplas restante al conjunto de entrenamiento, siempre y cuando no se perdiera la partición léxica. Éste algoritmo consta de aplicar un *Breadth-first search* (BFS) sobre el núcleo y nos dio buenos resultados en cuanto a la pérdida de tuplas y dimensiones de los conjuntos, quedando como la versión final utilizada

Como se explicó previamente la sección 4.5.3, el algoritmo toma el nodo más conectado como raíz y aplica el algoritmo BFS a partir de él, añadiendo aristas y nodos a la lista de visitados hasta completar la cantidad de antónimos necesaria en test. Luego estos nodos y aristas son desconectados del resto del grafo, y las tuplas representadas por ellas son asignadas al conjunto de test. Aquellas aristas que van de un nodo visitado a un nodo no visitado se descartan, y no serán utilizadas en ningún conjunto. Este procedimiento se repite para armar validación y las aristas que siguen siendo parte del grafo luego pasan a ser el conjunto de entrenamiento.

Esta página ha sido intencionalmente dejada en blanco.

Apéndice B

Espacios de búsqueda del random search

Inicialmente el espacio de búsqueda utilizado para los *random search* consistió en los siguientes elementos, eliminando aquellos que no aplican para el modelo *concat*:

- Cantidad de capas de la red neuronal en el rango de $[1, 4] \in N$.
- Tamaño de las capas $[100, 500] \in N$. Un valor para cada capa es seleccionado independientemente.
- tasa de aprendizaje (*Learning rate*): uno dentro de 1e-3, 1e-4, 1e-5
- Función de activación entre capas: una de [ReLU, Tanh, Sigmoid]
- *Batch size*, uno de: 64, 128, 512, 1024, 1548, 2048
- Márgenes, utilizados para el *constrasting loss* en los modelos siamesa y parasiamesa, y el valor *acceptance* utilizado para la clasificación. Estos valores se toman del rango $[0{,}01, 0{,}02, ..., 5]$. (No aplica a modelo concat)
- Estrategia: uno de *strict*, *lenient*, *acceptance*. (No aplica a modelo concat.)

Al analizar los resultados de las ejecuciones notamos una gran cantidad de instancias, de los modelos de tipo siamesa y parasiamesa, en las que las medidas F1, *accuracy* y *recall* eran muy bajas, y en algunos casos llegaban a ser 0. Para estas ejecuciones se observaron puntos comunes en cuanto a los parámetros que se habían utilizado. Se descubrió que en la mayoría de los casos la función de activación sigmoide generaba muy bajo desempeño. También se notó que con valores altos de *batch size* los resultados eran mejores, así como al utilizar *acceptance* como estrategia de clasificación, y al utilizar márgenes más altos. Otra notaciones realizadas fue que las tasas de aprendizaje más altas fueron menos eficaces y los tamaños de entrada de las capas estaban algo acotadas.

Apéndice B. Espacios de búsqueda del random search

Utilizando las observaciones anteriormente mencionadas se decidió mejorar nuestro espacio de búsqueda eliminando o cambiando opciones de los hiperparámetros para obtener una mayor cantidad de buenas combinaciones, e intentar llegar a mejores resultados.

El espacio de búsqueda final utilizado para los modelos de tipo siamesa y parasiamesa fue el siguiente:

- Cantidad de capas de la red neuronal en el rango de $[1, 4] \in N$.
- Tamaño de las capas $[100, 700] \in N$. Un valor para cada capa es seleccionado independientemente.
- tasa de aprendizaje (*Learning rate*): uno dentro de 1e-2, 1e-3, 1e-4
- Función de activación entre capas: una de [ReLU, Tanh]
- *Batch size*, uno de: 256, 512, 1024, 2048, 4096
- Márgenes, utilizados para el *constrasting loss* en los modelos siamesa y parasiamesa, y el valor *acceptance* utilizado para la clasificación. Estos valores se toman del rango $[0{,}01, 0{,}02, ..., 7]$.
- Estrategia: dejó de ser un hiperparámetro, quedando fijo en "acceptance"

Apéndice C

Gráficas de mejores modelos con simétricos

En las figuras C.2 C.3, C.4 y se presenta la evolución del entrenamiento de los mejores modelos entrenados con la versión simétrica de los conjuntos. No se realiza un análisis de las mismas debido a que son extremadamente similares a las gráficas presentadas en la sección 5.2.2.1.

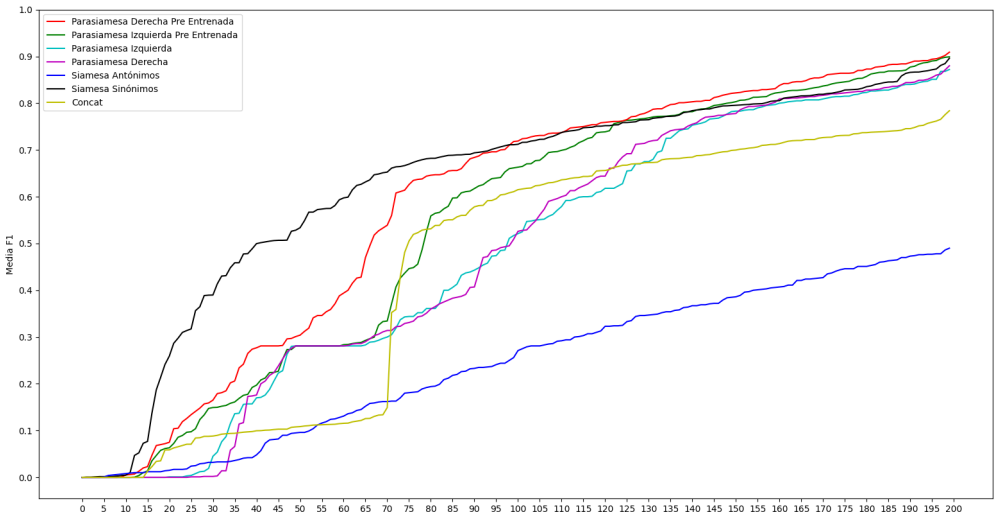

Figura C.1: Medida F1 en el conjunto de validación, de diferentes ejecuciones del *random search* en la partición *random* con simetricos

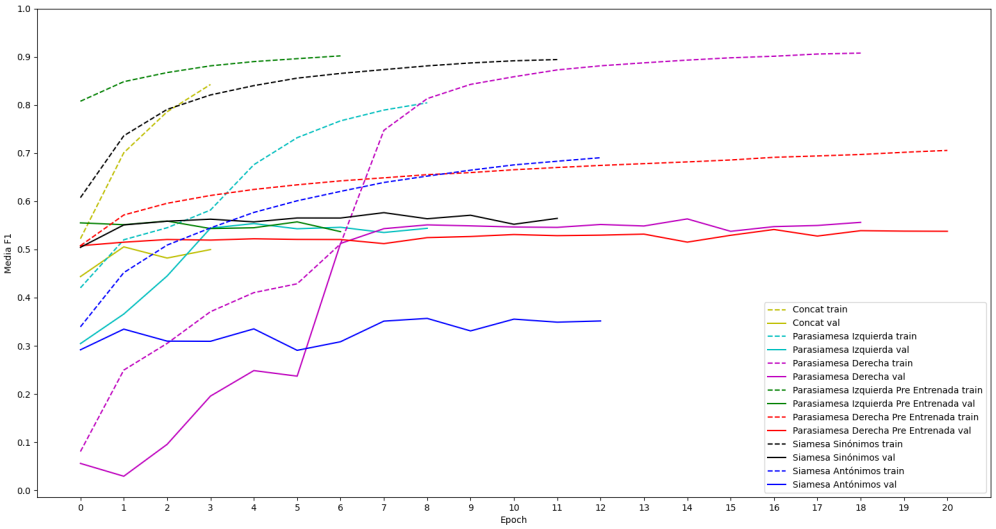

Figura C.2: Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición basada en *Shwartz*

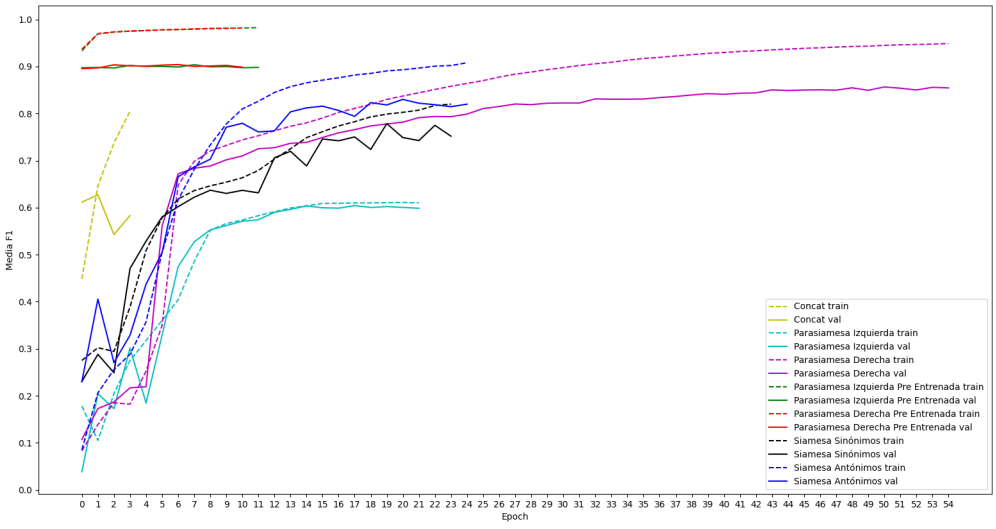

Figura C.3: Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición basada en grafo

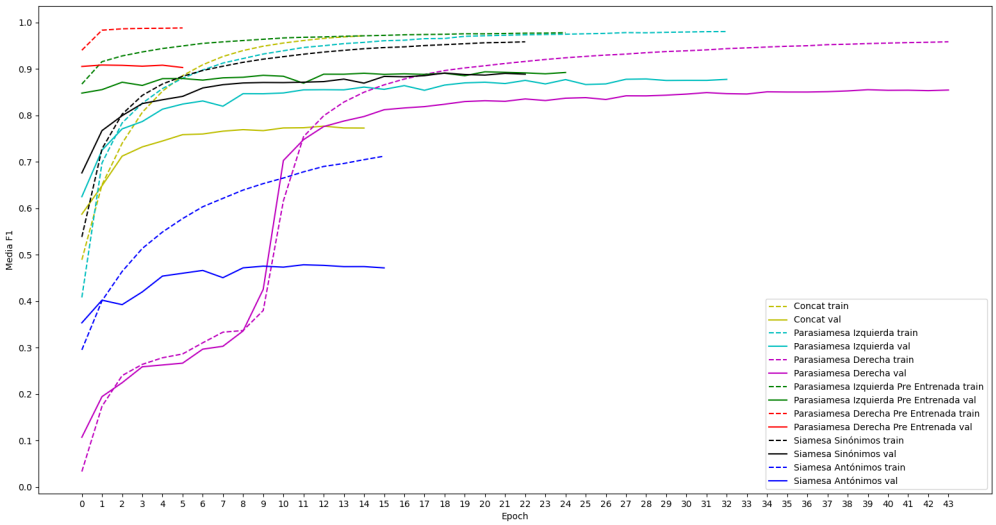

Figura C.4: Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición *random*

Esta página ha sido intencionalmente dejada en blanco.

Referencias

Muhammad Asif Ali, Yifang Sun, Xiaoling Zhou, Wei Wang, and Xiang Zhao. Antonym-synonym classification based on new sub-space embeddings. In *AAAI*, 2019.

Pieter Swart Aric Hagberg, Dan Schult. Networkx, July 2005. URL `https://github.com/networkx/networkx#readme`.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 2006. ISBN 978-0387-31073-2.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*, 2016.

Cristian Cardellino. Spanish Billion Words Corpus and Embeddings, August 2019. URL `https://crscardellino.github.io/SBWCE/`.

Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, aug 2016. doi: 10.1145/2939672.2939785. URL `https://doi.org/10.1145%2F2939672.2939785`.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960. doi: 10.1177/001316446002000104. URL `https://doi.org/10.1177/001316446002000104`.

D. A. Cruse. *Lexical Semantics*. Cambridge University Press, 2010. ISBN 0-521-27643-8.

Mathias Etcheverry and Dina Wonsever. Unraveling antonym's word vectors through a Siamese-like network. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3297–3307, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1319. URL `https://aclanthology.org/P19-1319`.

Ana Fernández-Montraveta, Gloria Vázquez, and Christiane Fellbaum. The spanish version of wordnet 3.0, 09 2008.

Zellig S. Harris. Distributional structure. *¡i¿WORD¡/i¿*, 10(2-3):146–162, 1954. doi: 10.1080/00437956.1954.11659520. URL https://doi.org/10.1080/00437956.1954.11659520.

Marti A Hearst. Automatic acquisition of hyponyms from large text corpora. In *COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics*, 1992.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997. doi: 10.1162/neco.1997.9.8.1735.

Harold Hotelling. Relations between two sets of variates. *Biometrika*, 28(3/4):321–377, 1936. ISSN 00063444. URL http://www.jstor.org/stable/2333955.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning, 2020. URL https://arxiv.org/abs/2004.11362.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

Dimitrios Kouzis-Loukas. *Learning Scrapy*. Packt Publishing Ltd, 2016.

Omer Levy, Steffen Remus, Chris Biemann, and Ido Dagan. Do supervised distributional methods really learn lexical inference relations? In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 970–976, 2015.

Richard Liaw, Eric Liang, Robert Nishihara, Philipp Moritz, Joseph E Gonzalez, and Ion Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

George A. Miller. Wordnet: A lexical database for english, nov 1995. ISSN 0001-0782. URL https://doi.org/10.1145/219717.219748.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill Science/Engineering/Math, 1997. ISBN 0070428077.

Nikola Mrkšić, Ivan Vulić, Diarmuid Ó Séaghdha, Ira Leviant, Roi Reichart, Milica Gašić, Anna Korhonen, and Steve Young. Semantic specialisation of distributional word vector spaces using monolingual and cross-lingual constraints, 2017. URL https://arxiv.org/abs/1706.00374.

Kim Anh Nguyen, Sabine Schulte im Walde, and Ngoc Thang Vu. Integrating distributional lexical contrast into word embeddings for antonym-synonym distinction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 454–459, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-2074. URL `https://aclanthology.org/P16-2074`.

Kim Anh Nguyen, Sabine Schulte im Walde, and Ngoc Thang Vu. Distinguishing antonyms and synonyms in a pattern-based neural network. *arXiv preprint arXiv:1701.02962*, 2017.

Carita Paradis. *Good, better and superb antonyms: a conceptual construal approach*, volume 3, pages 385–402. Charles University, 2010. ISBN 978-80-7308-290-1.

Carita Paradis, Caroline Willners, and Steven Jones. Good and bad opposites: using textual and experimental techniques to measure antonym canonicity. *The Mental Lexicon*, pages 380–429, 2009. ISSN 1871-1340.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

Sabine Schulte im Walde and Maximilian Köper. Pattern-based distinction of paradigmatic relations for german nouns, verbs, adjectives. In *Language Processing and Knowledge in the Web*, pages 184–198. Springer, 2013.

Vered Shwartz, Yoav Goldberg, and Ido Dagan. Improving hypernymy detection with an integrated path-based and distributional method. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2389–2398, Berlin, Germany, August 2016. Association

for Computational Linguistics. doi: 10.18653/v1/P16-1226. URL https://aclanthology.org/P16-1226.

A. M. TURING. I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, LIX(236):433–460, 10 1950. ISSN 0026-4423. doi: 10.1093/mind/LIX.236.433. URL https://doi.org/10.1093/mind/LIX.236.433.

Ivan Vulić. Injecting lexical contrast into word vectors by guiding vector space specialisation. In *Proceedings of The Third Workshop on Representation Learning for NLP*, pages 137–143, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/W18-3018. URL https://aclanthology.org/W18-3018.

Zhipeng Xie and Nan Zeng. A mixture-of-experts model for antonym-synonym discrimination. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pages 558–564, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-short.71. URL https://aclanthology.org/2021.acl-short.71.

Índice de tablas

4.1. Tuplas clasificadas de forma distinta debido a la acepción utilizada por el anotador. 32
4.2. Tuplas clasificadas de forma distinta debido a la connotación interpretada por el anotador. 32
4.3. Cantidades de los conjuntos en su versión sin simétricos 35

5.1. Mejores combinaciones de hiperparámetros encontradas al utilizar la partición *random* sin simétricos 44
5.2. Mejores combinaciones de hiperparámetros encontradas al utilizar la partición con separación léxica basada en Shwartz sin simétricos. 44
5.3. Mejores combinaciones de hiperparámetros encontradas al utilizar la partición generada con el algoritmo basado en el grafo sin simétricos. 45
5.4. Métricas obtenidas al evaluar en conjuntos de *test* simetrizado los modelos entrenados con conjuntos sin simétricos ni reflexivos . . . 47
5.5. Métricas obtenidas al evaluar en conjuntos de *test* simetrizado los modelos entrenados con conjuntos de entrenamiento y validación también simetrizados. 48

Esta página ha sido intencionalmente dejada en blanco.

Índice de figuras

2.1. Gráfica de la función *sigmoid* 5
2.2. Ejemplo de neurona artificial. 6
2.3. Funciones de activación populares.[1] 7
2.4. Red neuronal multicapa feed-fordward 8
2.5. Red siamesa 9
2.6. Captura de pantalla de la web de WordNet con *synsets* para la palabra *hot*. [2] 17

3.1. Árbol sintáctico de dependencias para la oración "My old village has been provided with the new services" donde el camino de "old" a "new" se encuentra marcado de color rojo y esta formado por (old-village-provided-with-services-new). Imagen extraída de Nguyen et al. (2017) 20
3.2. Imagen extraída de Nguyen et al. (2017) donde se ilustra el modelo AntSynNET presentado por los autores. 22
3.3. Estructura interna de una parasiamesa izquierda y derecha respectivamente, imagen basada en Etcheverry and Wonsever (2019) . . 26

4.1. Grafo de la relación de **antonimia** generado a partir de dataset. Se puede observar que el grafo cuenta con una gran componente conexa (16651 nodos) la cual esta rodeada por 298 componentes disconexas entre si compuestas de entre 2 y 11 nodos 33
4.2. Grafo de la relación de **sinonimia** generado a partir del dataset. Al igual que la relación de antonimia, en la sinonimia se observa un núcleo conteniendo 44576 nodos mientras que el conjunto de componentes disconexas esta conformado por 9770 los cuales contienen entre 2 y 20 nodos 33
4.3. Recorrida BFS de un grafo.[3] 37

5.1. Diagrama de la red neuronal correspondiente al modelo concat. . . 40
5.2. Medida F1 en el conjunto de validación, de diferentes ejecuciones del *random search* en la partición *random* 43
5.3. Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición *random* . . . 46

5.4. Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición basada en grafo 47
5.5. Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición basada en *Shwartz* 48

A.1. Ejemplo de articulación de un grafo y las componentes conexas generadas si el mismo es eliminado. 54
A.2. Ejemplo de arista de corte de un grafo y las componentes conexas generadas si la misma es eliminada. 54

C.1. Medida F1 en el conjunto de validación, de diferentes ejecuciones del *random search* en la partición *random* con simetricos 59
C.2. Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición basada en *Shwartz* 60
C.3. Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición basada en grafo 60
C.4. Evolución de Medida F1 (medida en los conjuntos de entrenamiento y validación) durante el entrenamiento de la ejecución encontrada en el *random search* para cada modelo en la partición *random* . . . 61

Esta es la última página.
Compilado el domingo 6 noviembre, 2022.
http://fing.edu.uy/es/inco