Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Un modelo de redes neuronales artificiales inspirado en la semántica cognitiva

Tesis presentada para optar al título de Doctor de la Universidad de Buenos Aires en el área de Ciencias de la Computación

Lado, Gustavo Alberto

Buenos Aires, 2022

Director: Prof. Dr. Enrique Carlos Segura

Consejero de Estudios: Prof. Dr. Julio Jacobo Berlles

Lugar de trabajo: Dpto de Computación, FCEyN, UBA

Resumen

La extracción y representación eficiente de información a partir del conocimiento expresado en lenguajes naturales es de suma importancia teniendo en cuenta la gran cantidad de saber que existe en esta forma. Sin embargo, estos problemas han probado ser de difícil tratamiento a pesar de haber sido empleadas variadas estrategias a lo largo de los años.

Originalmente las técnicas de inteligencia artificial para el procesamiento del lenguaje natural estaban basadas en la semántica estructuralista, por lo que requerían del análisis sintáctico de las sentencias, y la clasificación en categorías semánticas de las palabras, entre otros. Actualmente los modelos basados en redes neuronales artificiales pueden obtener muy buenos resultados y no requieren del mismo tipo de preprocesamiento sintáctico pero tienden a poseer una alta complejidad y costo de entrenamiento.

En este trabajo se propone un nuevo tipo de modelo alternativo, también basado en redes neuronales artificiales e inspirado en la semántica cognitiva, para convertir información secuencial en vectores de dimensión fija.

Este modelo fue utilizado en la codificación de tanto palabras como sentencias. En el caso de las palabras se pudo demostrar su capacidad para representar información morfológica y robustez ante variaciones asociadas a distintos tipos de errores. Para las sentencias también se pudo verificar la correlación semántica entre representaciones obtenidas por el modelo y evaluaciones hechas por humanos sobre datos provenientes de fuentes reales, como también otras estimaciones realizadas para datos artificiales.

Así mismo, el modelo propuesto no solo brinda una alternativa más sencilla evitando las desventajas mencionadas inicialmente, sino que además permite superar algunos inconvenientes aún existentes en las técnicas más avanzadas y facilita la integración con otras estrategias de aprendizaje automático.

Palabras clave: Redes neuronales artificiales, procesamiento del lenguaje natural, aprendizaje profundo, semántica cognitiva

An artificial neural network model inspired by cognitive semantics

Abstract

The extraction and representation of information expressed in natural languages in an efficient manner is of utmost importance considering the significant amount of knowledge that exists in this form. However, these problems have proven to be difficult to handle despite the use of various strategies over the years.

Originally, artificial intelligence techniques for natural language processing were based on structuralist semantics, so they required the syntactic analysis of sentences, the classification of words into semantic categories and so forth. Currently, models based on artificial neural networks can obtain very good results and do not require the same type of syntactic preprocessing, but tend to have a high complexity and training cost.

In this work, a new type of alternative model, also based on artificial neural networks and inspired by cognitive semantics, is proposed to convert sequential information into fixed-dimensional vectors.

This model was used for the encoding of both, words and sentences. In the case of words, it was able to demonstrate its ability to represent morphological information and robustness to variations associated with different types of errors. For sentences, it was possible to verify the semantic correlation between representations obtained by the model and evaluations made by humans on data from real sources, as well as other estimations made for artificial data.

Besides, the proposed model not only provides a simpler alternative, avoiding the disadvantages mentioned initially, but also it allows to overcome some issues remaining in the most advanced techniques and, it facilitates the integration with other machine learning strategies.

Keywords: Artificial neural networks, natural language processing, deep learning, cognitive semantics

Índice general

1. Introducción

1.1. Antecedentes y Motivación

Para una gran variedad de dominios de problemas las técnicas basadas en aprendizaje automático son generalmente las que pueden ofrecer los mejores resultados. Sin embargo, la mayoría de estas técnicas suelen representar a los datos de entrada y a las respuestas como vectores de dimensión fija. Esto presenta una barrera para tratar problemas cuya naturaleza es fundamentalmente secuencial [26].

Por ejemplo, en el caso de las redes neuronales, una de las primeras soluciones adoptadas fue la de utilizar la entrada como una ventana de dimensión fija que pudiera ser desplazada sobre la secuencia de datos. Posteriormente se desarrollaron métodos de aprendizaje como Back-Propagation Through Time [107, 103] y Real Time Back-Propagation [127] que brindaron un tipo de solución más amplia. Así mismo también se introdujeron modelos como las Simple Recurrent Neural Networks [31] y las Recursive Auto-Asociative Memory [91] que presentaron las primeras soluciones reales, aunque limitadas, para trabajar con datos secuenciales. La combinación de estas y otras estrategias culminó en la creación de modelos como las Long Short-Term Memory (LSTM) [48] y las Gated Recurrent Units (GRU) [25], que actualmente no solo son capaces de trabajar eficazmente con datos secuenciales sino que también pueden aprender relaciones temporales de largo plazo. Esto no significa que se trate de un problema cerrado y aún se siguen investigando otro tipo de alternativas. Por ejemplo modelos como las WaveNet [87] y VDCNN [20] utilizan redes convolucionales en 1D.

Pero por supuesto que en la actualidad el mayor área de interés para este tipo de modelos está enfocado en el Procesamiento de Lenguajes Naturales. Los lenguajes naturales presentan además sus propios desafíos. Por ejemplo, las oraciones bien construidas deben obedecer una serie de reglas gramaticales, ortográficas, etc; pero además estas reglas también suelen contar con numerosas excepciones. Aún más, dentro del conjunto de todas las construcciones sintácticamente correctas del lenguaje, solo una pequeña porción son también semánticamente válidas. Y esto se debe a que están restringidas por el dominio que están describiendo, ya que el lenguaje es utilizado para describir un dominio externo que cuenta con sus propias reglas y excepciones. Además, a pesar de que la cantidad de información es abundante, suele ser difícil incorporarla en un estrategia de aprendizaje porque, por lo general, los datos están mayormente desorganizados y sin clasificar. Sin embargo, todas estas condiciones hacen que el problema sea ideal para probar la capacidad de un modelo orientado al procesamiento de datos secuenciales.

Originalmente los métodos adoptados por la inteligencia artificial para tratar el problema estaban basados en principios derivados de la semántica estructuralista [14, 34, 105]. En particular, dos de estos principios utilizados comúnmente eran el de Características Semánticas [105, 106], en donde se asume que las categorías son entidades discretas caracterizadas por un conjunto de propiedades que son compartidas por todos sus miembros, y el Principio de Composicionalidad [34, 14], por el que se cree que el significado del todo es función del significado de las partes y cómo éstas son combinadas sintácticamente.

Muchas técnicas que siguen vigentes en la actualidad como Word Tokenización, Part-of-Speech (PoS) tagging, Name Entity Recognition (NER), Semantic Role Labeling, etc. están aún apoyadas en estas hipótesis de trabajo [3, 72, 4, 115].

La composicionalidad es una característica deseable, pero no está claro que esté exclusivamente ligada a la sintaxis. La categorización también es deseable, pero no está claro que cada característica forme una entidad completamente discreta. Como consecuencia de esto, los sistemas basados en estos principios [113, 13] sufren las desventajas de requerir de un análisis previo de los datos, estar ligados a la gramática de un lenguaje en particular y ser poco robustos antes errores o nueva información, entre otras.

Un tipo de estrategia alternativa que surgió para evitar estos inconvenientes es la llamada Bolsa-de-Palabras (BoW, del inglés Bag-of-Words) [128]. Esta consiste en representar a las sentencias marcando la presencia de palabras en un vector cuya dimensión es igual al tamaño de un vocabulario elegido, por lo general dependiente del conjunto de datos. De esta forma no solo se pueden evitar los inconvenientes anteriores sino que también es posible mejorar sus resultados. Por ejemplo, incorporando información estadística con estrategias como Term Frequency-Inverse Document Frequency (TF-IDF) [96], o explotando las correlaciones en la ocurrencia de palabras con métodos como el Latent Semantic Analysis (LSA) [30, 64] o Latent Dirichlet Allocation (LDA) [7]. Sin embargo los resultados que ofrecen estás técnicas están intrínsecamente limitados por la cantidad de información que se conserva en este tipo de representación. Lo más notable es, por supuesto, que se está descartando el orden de la palabras, pero además también suelen omitirse las palabras muy comunes, llamadas Stop Words, u otros términos que aparezcan muy rara vez [124].

Un cambio de paradigma permitió obtener resultados prometedores a partir del uso de redes recurrentes como las LSTM para crear modelos de lenguaje. Esta estrategia consiste en usar al modelo neuronal para que aprenda a predecir la siguiente palabra en la secuencia [5]. En su forma más sencilla se puede representar a las palabras individuales dentro de una sentencia en forma similar a las BoW, con un vector de dimensión igual al tamaño del vocabulario con todos sus valores en cero salvo un uno en la posición que corresponda a la palabra indicada. Este tipo de codificación es llamada One-Hot Vector.

Sin embargo, progresos más significativos se hicieron al cambiar las representaciones de las palabras de One-Hot por los llamados word-embeddings. Estos permiten representar a las palabras con vectores de dimensión menor y valores acotados pero continuos en lo que se conoce como un espacio de representación semántica, esto es, palabras con significados similares obtienen representaciones cercanas dentro de este espacio. Para

obtener estos word-embeddings las primeras técnicas fueron las llamadas word2vect y consisten en dos variantes, continuous bag-of-words (CBOW) y continuous skip-gram [77, 79]. Ambas se basan en la idea de tomar como codificación la activación en la capa oculta de una red neuronal que aprenda a hacer predicciones alrededor de una palabra y la ventana de palabras que la rodean. Estas mismas ideas fueron mejoradas posteriormente con técnicas similares como Global Vectors (GloVe) [88].

Esto, sin embargo, introdujo otros problemas. Por ejemplo los homónimos pueden terminar teniendo una representación para una sola palabra con un vector que sea un compromiso entre los distintos significados posibles, sin representar particularmente bien a ninguno. Otro problema que se suele presentar es con palabras desconocidas o mal escritas, que no formaban parte del vocabulario original (out-of-vocabulary), o cuando se quiere trasladar un modelo que fue entrenado en un dominio a otro distinto, en donde algunas palabras pueden cambiar de significado.

En parte estos problemas fueron tratados con la introducción de modelos como ELMo [89], que produce word-embeddings para cada palabra dependiendo del contexto en el que se encuentra utilizando una red LSTM bidirenccional (BiLSTM) con entrada a nivel carácter, o fastText [8] que también incorpora información de sub-palabras en la forma de n-gramas. Y aunque también existen otras estrategias para representar la entrada utilizando una codificación por sub-palabras, como los character level embeddings [134, 58], o basadas en grupos de bytes [112, 37], la mayoría de los modelos actuales aún sigue dependiendo de un vocabulario predeterminado.

La incorporación de embeddings y bidireccionalidad permitió mejorar los resultados de las redes recurrentes de compuertas básicas, pero una efectividad aún mayor pudo alcanzarse gracias a la incorporación de otra estrategia más, los mecanismos de Atención. Estos permiten ponderar la relevancia de cada palabra dentro de la secuencia de acuerdo a la tarea. Estos mecanismos son tan efectivos que dieron lugar a la creación de un nuevo tipo de modelo llamado Transformer [122], que pudo producir resultados igualmente buenos utilizando solo atención y sin necesidad de conexiones recurrentes. Actualmente modelos como BERT [24], que utiliza un mecanismo de auto-atención, o la familia de modelos GPT [94], pueden aplicarse a una gran diversidad de tareas con excelentes resultados.

Sin embargo, estos modelos tienen una complejidad y costo computacional muy altos. Modelos basados en Transformes tienen un costo computacional $O(N^2)$ en relación a la longitud de la secuencia. Alternativas como Reformers [61] con $O(N \ln N)$, o BigBird [131] con $O(N)$, como también XLNet [130] o RoBERTa [70] intentan ser optimizaciones menos costosas con una efectividad equivalente. Pero modelos como GPT-3, que tiene 175 billones de parámetros, son tan costosos de entrenar que llegaron a producir preocupaciones acerca de su impacto ambiental o problemas éticos por las limitaciones de acceso. Además, a pesar de que los mecanismos de atención pueden dar una idea del funcionamiento interno de modelos como BERT, las razones para su nivel de desempeño no están completamente claras, ya que no existe una teoría lingüística que las respalde.

Todos estos modelos son muy capaces en una gran diversidad de tareas, pero ninguno está particularmente orientado a producir codificaciones vectoriales de sentencias

que puedan ser integradas fácilmente con otras estrategias de aprendizaje automático. Para esto fueron desarrolladas otras técnicas análogas a los word-embeddings llamadas sentence-embedding. Las más sencillas de estas técnicas consisten en promediar o concatenar los word-embeddings dentro de la sentencia, y aún con una estrategia tan elemental pueden conseguirse resultados aceptables [80]. De forma análoga, un método llamado SkipThought o Doc2Vec [60], está basado en los mismos principios que word2vect, utilizando las sentencias de contexto para hacer una auto-codificación con una red GRU. Y hasta también ofrece dos tipos de variantes, Paragraph Vector Distributed Memory (PVDM) y Paragraph Vector Distributed Bag of Words (PVFOBW) [66]. Otros modelos también utilizan estrategias similares a los word-embeddings pero a nivel sentencia, como SentenceBERT [100] que usa dos transformers gemelos entrenados en tareas de similitud semántica o clasificación. El sentence-embedding finalmente es construido mediante una etapa de pooling. O InferSent [19], que usando word-embeddings provenientes de GloVe o fastText codifica sentencias mediante una red GRU o BiLSTM, y entrena al modelo utilizando concatenaciones de las sentencias codificadas, su diferencia absoluta, y su producto por elemento, con un clasificador entre tres categorías: entail, neutral, o contradict. Finalmente el modelo Universal Sentence Encoder (USE) [12] introduce una técnica interesante llamada Deep Averaging Network (DAN) [54] que consiste en producir unigramas y bigramas de los embeddings, y promediarlos para que estos vectores puedan ser pasados a su vez a otra red u otra etapa similar para ser recodificados.

Otro aspecto importante es que, aunque originalmente las hipótesis de trabajo de la semántica estructuralista fueron adoptadas y utilizadas ampliamente, en la actualidad son usadas casi exclusivamente en combinación, pero de manera poco consistente, con hipótesis de trabajo provenientes de otras teorías lingüísticas como la semántica cognitiva [119, 33].

En contraste con la semántica estructuralista, el enfoque de la semántica cognitiva rechaza la forma tradicional de modularización de la lingüística en fonología, sintaxis, pragmática, etc. y explica el significado principalmente en términos de la forma en que las percepciones son agrupadas en categorías conceptuales [119]. Así, el lenguaje y la cognición se consideran inseparables, porque supone que la estructura de las categorías lingüísticas refleja la estructura de las categorías conceptuales.

Todo esto no significa que los avances producidos durante los últimos años en el área de Procesamiento del Lenguaje Natural no hayan sido de una enorme magnitud e importancia, sin embargo aún existen algunos problemas y limitaciones que deben ser superadas. En particular algunas de estas desventajas consisten en que la complejidad y costo computacional del entrenamiento de los modelos más avanzados es muy alto, y que casi todas las estrategias dependen de las mismas técnicas, basadas en modelos de lenguaje orientados a la predicción de la siguiente palabra, word-embeddings precomputados con limitaciones a un vocabulario fijo, y sin ser especialmente aptos para producir sentence-embeddings.

1.2. Objetivos de la Investigación

Teniendo en cuenta que extraer información de lenguajes naturales sigue siendo de mayor importancia, que las redes neuronales artificiales son capaces de conseguir buenos resultados en este área, que los modelos actuales tienden a ser demasiado complejos y que no están orientados a producir codificaciones de sentencias que puedan ser integradas con otras estrategias de aprendizaje automático, y que sería deseable un soporte más consistente en una teoría lingüística, el objetivo general de este trabajo es:

- Proponer un nuevo tipo de modelo de redes neuronales artificiales inspirado por hipótesis de la semántica cognitiva que permita codificar secuencias de datos de longitud variable en vectores de dimensión fija.

Para poder llevar adelante esta meta se perseguirán también los siguientes objetivos específicos:

- Elaborar nuevas técnicas para mejorar el entrenamiento y codificación de datos con este tipo de modelos.
- Analizar la correlación entre la codificación producida por el modelo y la información de la secuencia original.
- Evaluar la capacidad del modelo para producir codificaciones que representen información semántica cuando se está trabajando con lenguajes naturales.

Si bien en este trabajo se presentan métodos novedosos basados en redes neuronales, no se pretende competir directamente con la variedad de tareas y nivel de desempeño de otros modelos más avanzados, sino que la principal contribución de estas técnicas sería ofrecer una alternativa más sencilla, no necesariamente más eficaz, pero si más eficiente, que eventualmente también pueda servir como base para futuros estudios.

1.3. Fundamento Teórico

La semántica cognitiva [119, 33] es un enfoque interdisciplinario para el estudio del significado y de la mente basado en los hallazgos de la psicología cognitiva. Esta teoría lingüística considera al lenguaje como uno de los principales sistemas cognitivos y estudia los procesos involucrados en la manipulación gramatical. Para esto asume que la semántica puede ser dividida principalmente en dos aspectos, la representación del conocimiento y la construcción del significado [120, 33].

Tradicionalmente con la semántica estructuralista la construcción del significado dependía del Principio de Composicionalidad y la representación del conocimiento de las Características Semánticas. Un enfoque basado en la semántica cognitiva implica entonces la adopción de unas hipótesis de trabajo distintas a estas.

En reemplazo del Principio de Composicionalidad se propone adoptar la Hipótesis Distribucional, según la cual, las palabras que suelen ocurrir en los mismos contextos tienden

a tener el mismo significado [43, 109, 74]. Y en lugar de las Características Semánticas se propone utilizar la Teoría de Prototipos, que dice que, según sus características, los miembros pertenecen de forma graduada a categorías, dependiendo de su similitud con un prototipo [63, 35].

La Hipótesis Distribucional brinda una justificación teórica para poder extraer significado del lenguaje considerándolo como sólo una secuencia de símbolos. Esto es, aprovechando las co-ocurrencias de palabras, sin necesitar de la composicionalidad sintáctica. La Teoría de Prototipos permite asumir que es posible representar el significado de forma vectorial. Esto es, utilizando tantas dimensiones como características se quiera considerar, y en donde las categorías pueden ser interpretadas como cúmulos[1] alrededor de los prototipos semánticos [76, 91, 116].

1.4. Justificación de la Propuesta

La redes neuronales artificiales son modelos que pueden adaptarse a un gran número de problemas. Generalmente poseen una arquitectura organizada en capas compuestas por un número variable de unidades neuronales que son activadas progresivamente desde una entrada hasta una salida (feedforward), en donde cada una de las unidades suele estar conectada con todas las unidades de la capa anterior (fully-connected) mediante valores llamados parámetros entrenables. Un conjunto de datos está formado por elementos que representan distintas instancias de un problema, en donde cada instancia puede estar compuesta por un patrón o vector de entrada y otro objetivo (supervised learning). La versatilidad de esta clase de modelos proviene de la libertad para la elección de distintas arquitecturas y la variedad de soluciones para modificar los parámetros entrenables a partir de diferentes conjuntos de datos.

Para poder adoptar las nuevas hipótesis de trabajo en un modelo de redes neuronales se propone utilizar una arquitectura basada en auto-codificadores apilados [41].

Los auto-codificadores[2] son un tipo de red neuronal artificial perteneciente a la categoría de memorias auto-asociativas, y generalmente consistente en tres capas entrenadas para que la respuesta de salida sea igual al estímulo de entrada, generando de esta forma una representación interna distribuida de los datos en la capa oculta.

Una clase particular de codificación distribuida es la llamada codificación dispersa, en donde solo una porción de unidades tiende a estar activa simultáneamente. Esta clase de codificación es ideal para el tipo de representación que se quiere adoptar con la Teoría de Prototipos [108, 16, 104].

Los auto-codificadores apilados, por otro lado, son una clase de arquitectura profunda[3] en la que la codificación interna producida por un auto-codificador en un nivel es pasada

[1] del inglés "clusters".
[2] del inglés "autoencoders".
[3] del inglés "Deep Architecture", en relación al Aprendizaje Profundo, en inglés "Deep Learning".

como entrada a otro auto-codificador en su nivel superior; que a su vez, pasará su codificación interna al siguiente, y así.

Haciendo que cada auto-codificador tome al menos un par de patrones de un nivel y lo codifique en uno solo para el siguiente permitiría representar a las distintas correlaciones que ocurran dentro de las secuencias en diferentes niveles de la arquitectura. Estas correlaciones pueden capturar la co-ocurrencia de palabras en distintos contextos que, según la Hipótesis Distribucional, es necesaria para poder construir significado [123, 92].

También se debe notar que la Hipótesis Distribucional habla de las palabras que tienden ocurrir en el mismo contexto, por lo tanto para que estas tendencias sean significativas será necesario trabajar con conjuntos de datos grandes.

Esto hace que una arquitectura profunda no solo sea conveniente, sino necesaria, ya que además de ofrecer flexibilidad en la elección de cantidad de capas, unidades, y parámetros entrenables, estas brindan escalabilidad y adaptabilidad para distintos tipos problemas, permitiendo realizar un entrenamiento más efectivo [111, 6, 17].

El uso de auto-codificadores además tiene algunas ventajas sobre otras soluciones existentes. Los auto-codificadores son modelos extensamente probados y estudiados, que no están sujetos a problemas potencialmente inesperados, como el gradiente de fuga[4]. También son más fácilmente paralelizables que las redes recurrentes para el entrenamiento, y el tipo de arquitectura con estos modelos apilados permite que el entrenamiento pueda ser realizado de forma progresiva.

Existen distintas variantes posibles dentro de los auto-codificadores, pero en casi todos los casos se busca cambiar el espacio de representación de los datos, generalmente a uno de dimensión menor, preservando la mayor cantidad de información posible. Esta representación ocurre en la capa oculta, y para obtener el tipo de codificación dispersa que se busca, suele ser necesario incorporar alguna restricción adicional sobre la función de transferencia o la función de costo [86, 93].

Como parte de este trabajo también se propone una variante de auto-codificador que incorpora una función de transferencia capaz de producir un tipo de codificación dispersa concebida para trabajar con datos que posean distintos grados de correlación entre sus variables e instancias. Esto facilitaría la representación de tanto regularidades como excepciones en los datos [62].

Si bien los auto-codificadores en general pueden ser entrenados más fácilmente que las redes recurrentes, la incorporación de las restricciones adicionales para obtener la codificación dispersa deseada hacen al proceso más demandante. Por esto, se propone además una nueva técnica de minimización de la función de costo que permite acelerar el entrenamiento [40].

Cabe destacar también que la incorporación de las nuevas hipótesis de trabajo no limitaría necesariamente al modelo al dominio del Procesamiento del Lenguaje Natural. Es decir que finalmente la arquitectura no incorpora ninguna restricción específica del lenguaje y puede trabajar con cualquier tipo de datos secuenciales.

[4] en inglés "vanishing gradient".

Teniendo en cuenta que no solo es posible considerar a las sentencias como secuencias de palabras, sino que a su vez las palabras también pueden ser consideradas como secuencias de letras, o símbolos en general, y dada la posibilidad de trabajar con secuencias de cualquier tipo, se propone la utilización del mismo tipo de arquitectura tanto para la codificación de palabras como de sentencias. Esto puede ser llevado a cabo utilizando un primer modelo que tome secuencias de letras, números, y otros símbolos y los codifique en vectores de dimensión fija. Un segundo modelo puede entonces utilizar estas representaciones como secuencias de palabras para que sean codificadas en nuevos vectores para las sentencias.

Esto tiene como beneficio adicional que no sea necesaria la incorporación de una etapa previa que capture las correlaciones semánticas entre palabras como los word-embeddings, ya que estas correlaciones serán codificadas naturalmente por las etapas intermedias del modelo de sentencias. La arquitectura permite además introducir un nivel de redundancia que hace que la representación no sea solo tolerante ante palabras nuevas o mal escritas, sino que también la hace robusta ante oraciones gramaticalmente incorrectas, mal formadas, o muy atípicas.

Finalmente el modelo propuesto también cuenta con la ventaja de no necesitar de datos etiquetados para su entrenamiento, ya que se trata de un tipo de modelo auto-supervisado. Pero de todas formas, si el tipo de problema o tarea lo requiere, sigue siendo posible hacer un entrenamiento supervisado final. Además, por su naturaleza auto-supervisada, la misma arquitectura puede ser utilizada para generar o decodificar secuencias, sin requerir de un entrenamiento o etapa posterior.

La mayor limitación de esta arquitectura es que, a pesar de poder trabajar con secuencias de longitud variable, impone una restricción en la longitud máxima que no suele estar presente en los modelos recurrentes. Sin embargo, en la práctica, para facilitar el entrenamiento, muchos modelos recurrentes también deben trabajar con una limitación similar que permita paralelizar el aprendizaje.

1.5. Metodología

En primer lugar se desea demostrar la eficacia de las técnicas de entrenamiento y codificación propuestas. Para esto se las comparará con técnicas comúnmente usadas en la actualidad para distintos tipos de problemas, incluyendo conjuntos de datos que simulen con varios grados de dificultad las condiciones que se espera encontrar en la codificación de secuencias.

Posteriormente se quiere estudiar el comportamiento general del modelo cuando las secuencias de datos provienen, en distintas formas, de lenguajes. En particular se desea analizar el tipo de información que las codificaciones son capaces de capturar con respecto a las secuencias originales. Para esto, como instancia inicial, se utilizará un conjunto de datos artificial generado a partir de una gramática libre de contexto que permite construir sentencias válidas dentro de un dominio específico.

La gramática fue construida de forma que cada sentencia generada tuviera también un vector asociado para representar su semántica. Estas representaciones semánticas son invariantes ante los distintos tipos de construcciones sintácticas de sentencias con igual significado. El objetivo es poder comparar la codificación producida por el modelo con esta representación, que sirve a modo de ground-truth, para determinar el grado de correlación entre ambas.

Esto sirve también para evitar los inconvenientes asociados a determinar la calidad de conjuntos de datos provenientes de ámbitos reales y la dificultad para dar una medida objetiva de algo tan subjetivo como la semántica del lenguaje.

Así mismo se estudiará el grado de representación semántica en la codificación de grupos de palabras a partir de la distancia vectorial. Si las palabras, a pesar de inicialmente haber recibido representaciones binarias aleatorias, reciben codificaciones similares cuando tienen significados similares, será posible visualizar agrupamientos mediante un dendrograma sobre la distancia de codificación [125].

Se desea también evaluar la robustez del modelo estudiando su comportamiento ante errores o nueva información. Para esto se recurrirá a la codificación de un vocabulario de palabras en inglés, evaluando cómo varia la distancia de codificación al introducir distintos tipos y cantidad de errores similares a los ortográficos y tipográficos en relación a palabras fuera del vocabulario. Así mismo también se estudiará la capacidad de codificar palabras fuera de vocabulario en relación a composicionalidad morfológica a partir de palabras conocidas.

Finalmente se desea estudiar la capacidad del modelo al ser utilizado con lenguajes naturales sobre datos reales. Para esto se utilizarán conjuntos de sentencias provistas para la evaluación conocida como Semantic Texual Similarity. Esto consiste en comparar el grado de similitud semántica entre pares de sentencias en inglés de acuerdo a la opinión mayoritaria de evaluadores humanos. Se analizará el coeficiente de correlación de Pearson entre estas y la medida de semejanza del coseno entre las codificaciones producidas por el modelo. Estas evaluaciones estándar serán comparadas con el desempeño de otros modelos que participaron en estas mismas tareas, teniendo en cuenta la relación con la cantidad de parámetros entrenables.

1.6. Organización y Contenidos de la Tesis

El trabajo fue organizado en capítulos enfocados en temas individuales, tratando de mantener un formato que permitiera la lectura de los mismos de forma relativamente independiente. El contenido general de estos capítulos puede ser dividido, a grandes rasgos, en dos partes. En la primera parte se trata con el desarrollo de resultados teóricos, fundamentalmente orientados a la especificación del modelo y las técnicas necesarias para su entrenamiento. En la segunda parte se presentan resultados empíricos apoyados en estas teorías, involucrando experimentos que exploran la codificación de información morfológica y semántica, gramáticas artificiales, y lenguajes naturales. A continuación

se presenta una breve reseña de cada capítulo, destacando la motivación principal para cada uno y los resultados más relevantes.

En este capítulo se plantea la motivación para este trabajo y la justificación teórica del modelo, en particular en relación a las necesidades presentes en la disciplina del Procesamiento del Lenguaje Natural y las limitaciones de las técnicas contemporáneas.

En el capítulo 2 se define la arquitectura del modelo, estableciendo que clase de resultados es posible obtener con el mismo. En esta parte se tratan las cuestiones del aprendizaje de secuencias, la capacidad de memoria para trabajar con grandes datos, y la influencia de la Hipótesis Distribucional.

En el capítulo 3 se trata con las distintas clases de codificación que pueden ser utilizadas, y cuales son las mejores formas de obtenerlas. Esto es fundamental con respecto a la representación del conocimiento y la Teoría de Prototipos.

En el capítulo 4, en relación a las técnicas para obtener codificaciones, se plantea la necesidad de un mejor método para el descenso por gradiente y se propone un algoritmo alternativo. En especial se estudia la eficacia de la nueva técnica para el problema de reducción de dimensión con mínima pérdida de información.

El capítulo 5 comprende la primera etapa experimental con el modelo. En esta se estudia la relación entre el significado de sentencias y la representación obtenida en su codificación. Para poder tener un control sobre las variaciones semánticas utilizadas se recurrió a una gramática libre de contexto capaz de generar sentencias válidas en castellano dentro de un dominio restringido.

En el capítulo 6 se investiga la capacidad para capturar información morfológica en la codificación a partir de un extenso vocabulario de palabras en idioma inglés. En particular se estudia la validez de dos propiedades deseables en la representación obtenida, y su importancia cuando se desea utilizar al modelo para producir secuencias.

El capítulo 7 es el capítulo final, y es en donde se verifica experimentalmente la capacidad del modelo para generar representaciones vectoriales sensibles a la información semántica de sentencias pertenecientes a un lenguaje natural. Para ello se estudia la relación entre representaciones de pares de sentencias con un grado conocido de similitud semántica entre ellas.

Para los modelos y técnicas mencionados se trata de dar una traducción pertinente al castellano siempre que sea posible. Pero además se incluye una referencia al nombre original en inglés como las abreviaciones más comunes.

2. Auto-codificadores Apilados y Aprendizaje de Secuencias

2.1. Introducción

Uno de los principales temas de interés en el área del aprendizaje automático es el procesamiento del lenguaje natural, pero a pesar de los excelentes resultados que se han obtenido, todavía existe una barrera que es difícil de superar. La mayoría de las técnicas de aprendizaje automático están diseñadas para trabajar con información instantánea, por lo general representada en forma de arreglos unidimensionales de valores como vectores, y el lenguaje natural siempre se presenta como información secuencial [26]. Ya sea que consideremos palabras como secuencias de letras, oraciones como secuencias de palabras, o documentos como secuencias de oraciones, en todos estos casos podemos pensar que la información se nos presenta como una secuencia de símbolos, y para usar de manera efectiva estas técnicas de aprendizaje automático necesitamos, de alguna manera, convertir esta información secuencial en una representación vectorial. Pero, ¿qué tipo de representación vectorial queremos obtener? [69, 73, 108]

Por ejemplo, es deseable que la representación vectorial esté directamente relacionada con los símbolos que componen la secuencia. No sólo indicando qué símbolos están presentes, o en qué cantidad, sino también en qué orden [113]. Idealmente, la representación vectorial tendrá suficiente información sobre la secuencia para que sea posible reconstruirla.

También es deseable que la representación vectorial obtenida tenga la dimensión más pequeña posible [47, 116]. Algún grado de redundancia puede ser aceptable para la corrección de errores, pero dada la naturaleza y las posibles aplicaciones de este método, cada dimensión adicional en la representación puede llevar una carga computacional en etapas posteriores.

Un punto importante para la representación vectorial es que debe existir un grado de uniformidad o consistencia entre estas y todas las secuencias válidas. Esto significa que secuencias en algún sentido similares deben tener representaciones de vectores similares, por lo que la distancia vectorial entre dos representaciones diferentes podría usarse para medir la similitud de las secuencias a las que corresponden.

Idealmente, las representaciones podrían ser tan consistentes como para permitir operaciones vectoriales con restricciones pero respondiendo a un cierto grado de composicionalidad [78, 114]. Por ejemplo, podría ser útil si se pudieran realizar operaciones

aritméticas en las representaciones de vectores para obtener una representación cercana al resultado de hacer las mismas operaciones en las secuencias originales.

Y finalmente, es deseable que el método sea efectivo. Esto es, una vez aceptadas algunas restricciones sobre lo que se considera una secuencia válida, por ejemplo, en el conjunto de símbolos posibles o en la longitud máxima, el entrenamiento debería poder realizarse de una manera eficiente, es decir, no demandar demasiado tiempo o recursos computacionales, y una vez entrenado, el modelo debería ser capaz de producir codificaciones de manera eficaz, es decir, con una grado aceptable de generalización sobre cualquier secuencia válida [67, 77].

2.2. Antecedentes y Motivación

Actualmente, existen varios métodos que ofrecen soluciones similares, pero ninguno de ellos cumple con todas las propiedades descriptas anteriormente. Algunas de las alternativas más utilizadas son las Bolsas de Palabras [124], que se pueden obtener fácilmente pero con una gran dimensionalidad y una gran pérdida de información sobre la secuencia, las Redes Neuronales Recurrentes Simples [31, 98] que pueden generar una buena representación en la capa oculta, pero no para grandes conjuntos de datos, u otras variantes de redes recurrentes como las Redes de Memoria a Largo-Corto Plazo [48], que son más difíciles de entrenar y no ofrecen una representación vectorial realmente única como la que se busca aquí.

Dentro de los modelos utilizados en la actualidad los únicos que ofrecen una solución cercana a la buscada, y que parecen compartir varias características con el modelo propuesto, son los basados en mecanismos de atención, particularmente los Transformers [122].

En primer lugar, en los dos tipos de modelo el procesamiento de secuencias se logra sin necesidad de conexiones recurrentes, además ambos implementan una arquitectura basada en autocodificadores apilados, y los mecanismos de autoatención también podrían parecer similares a las técnicas de aprendizaje competitivo para codificación dispersa.

Sin embargo, todas estas semejanzas son solo superficiales. Los autocodificadores utilizados por los Transformers están formados por varias etapas de procesamiento con funcionalidades específicas. La información del orden de las palabras es codificada mediante la adición de una función de posicionamiento directamente sobre las representaciones de las palabras en un espacio semántico [95] y, posteriormente, el mecanismo de autoatención depende del producto entre vectores de representación generados separadamente llamados Query, Key, y Value, sujetos a varias cabezas de lectura independientes [10].

La propuesta de este trabajo se basa en una idea originada por un modelo no tan conocido llamado Memoria Auto-Asociativa Recursiva (RAAM) [91] que es, en cierto modo, una generalización del modelo de Red Recurrente Simple de Elman [31]. Una red RAAM se compone de un autocodificador capaz de comprimir un par de patrones a solo uno de

menor dimensión. Estos nuevos patrones comprimidos pueden retroalimentarse de forma recursiva en la red, lo que le permite aprender a codificar y decodificar estructuras de datos complejas, como listas y árboles. Esto parece hacerlo ideal para trabajar en problemas de lenguaje donde la información está definida por gramáticas con una estructura intrínsecamente recursiva [113, 114].

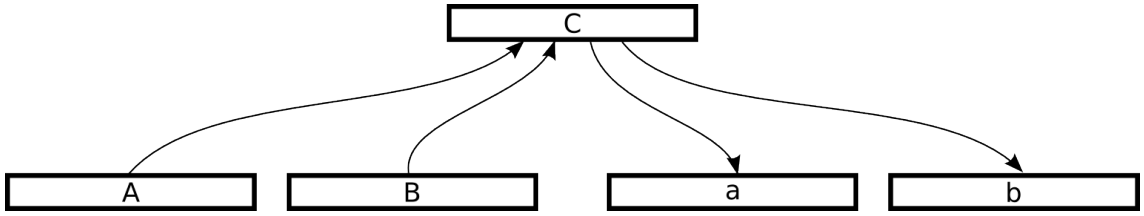

Figura 2.1.: Diagrama de una red RAAM.

En la Figura 2.1 se puede apreciar un diagrama de una red RAAM. Los patrones A y B serán codificados en el patrón C, indicado en el resto del texto como $[A, B] \rightarrow C$. A su vez el patrón C será decodificado en los patrones a y b, indicado como $C \rightarrow [a, b]$. Dado que existe un margen de error en la reconstrucción de los patrones A y B, sus equivalentes decodificados son indicados con la misma letra pero en minúscula (a y b) para destacar esta diferencia.

Por ejemplo para el árbol binario $((AB)(CD))$, mostrado en la Figura 2.2, en donde cada hoja es un patrón de tamaño fijo, esto tomaría los siguientes pasos. Primero, (AB) sería codificado en un patrón R_1 y (CD) sería codificado en un patrón R_2, independientemente del orden o inclusive en paralelo, y posteriormente (R_1R_2) sería codificado en un patrón R_3. El reconstructor debería decodificar estos patrones de tamaño fijo en patrones similares a sus partes, y determinar cuando estas partes deben también ser decodificadas. Este proceso podría ser aplicado recursivamente en forma descendente resultando en la reconstrucción del árbol original. En este caso, R_3 sería decodificado en (r_1r_2), r_1 en (ab) y r_2 en (cd).

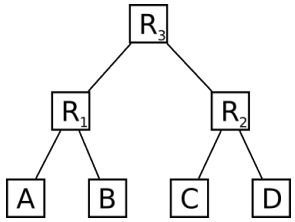

Figura 2.2.: Árbol ((A B) (C D)).

Mediante este mismo procedimiento una red RAAM puede ser entrenada para codificar y decodificar correctamente un conjunto de árboles con la propiedad adicional que los patrones resultantes formarán una representación distribuida de la estructura del árbol. Esto significa que estructuras similares entre sí serán codificadas en patrones similares entre sí, es decir, con una distancia vectorial menor entre ellos que con otros.

Sin embargo, este mismo mecanismo también tiene algunas desventajas; la codificación realizada en cada nodo del árbol está hecha por la misma red neuronal. Por consiguiente, todos los patrones deben tener las mismas dimensiones para poder ser re-alimentados recursivamente en el siguiente paso. Además esta única red debe aprender a codificar no solo los patrones que aparecen en las hojas, sino todos los que aparecen en los nodos internos. Cuando se usa una red única para contener toda la información para codificar y decodificar un conjunto de árboles, la capacidad del modelo está severamente limitada, y se vuelve menos robusta y más difícil de entrenar.

2.3. Descripción de la Arquitectura

Tratando de evitar las desventajas descriptas y mantener las propiedades deseables ya mencionadas, se propone la utilización de un modelo inspirado por las redes RAAM pero formado por una serie de autocodificadores organizados en etapas sucesivas. Con cada etapa encargada de aprender solamente los patrones correspondientes a su nivel.

La arquitectura estará formada por un *bloque* organizado en niveles en una serie de *etapas*. Cada *etapa* a su vez estará formada por un autocodificador con dos *capas* de entrada, una *capa* oculta, y dos *capas* de salida.

Los patrones codificados en un nivel, en lugar de ser realimentados a la misma etapa, son pasados mediante una conexión de copia a la etapa superior. De la misma forma, los patrones decodificados por una etapa no son realimentados a esta misma etapa sino que son pasados a la etapa inferior. Para construir los pares de patrones correspondientes a cada etapa se utiliza un tipo de conexión con retraso que se encarga de la coordinación.

De esta forma, cada etapa solo debe aprender las regularidades correspondientes a su nivel, es decir, el primer nivel aprende sobre las regularidades entre pares de símbolos, el siguiente sobre las regularidades entre pares de pares de símbolos, y así [108, 116, 123]. Esto aumenta la capacidad del modelo ya que la red en cada etapa solo debe aprender una parte del conjunto de patrones que antes debía aprender una sola red, y lo hace menos propenso a fallas.

En la Figura 2.3 se aprecia un diagrama del modelo. Las etapas están separadas por lineas de puntos. Cada etapa posee un autocodificador, similar al de una red RAAM, formada por las capas de entradas A y B, la capa oculta C y las capas de salida a y b. En estas capas se indica con un subíndice i la posición dentro de la secuencia de patrones y con un superíndice (n) la etapa a la que corresponde.

En la parte inferior, la capa I_i indica el patrón en la posición i de la secuencia de entrada. La secuencia de entrada estará definida por $I = [I_0, I_1, ...I_L]$ en donde cada I_i representa a un símbolo.

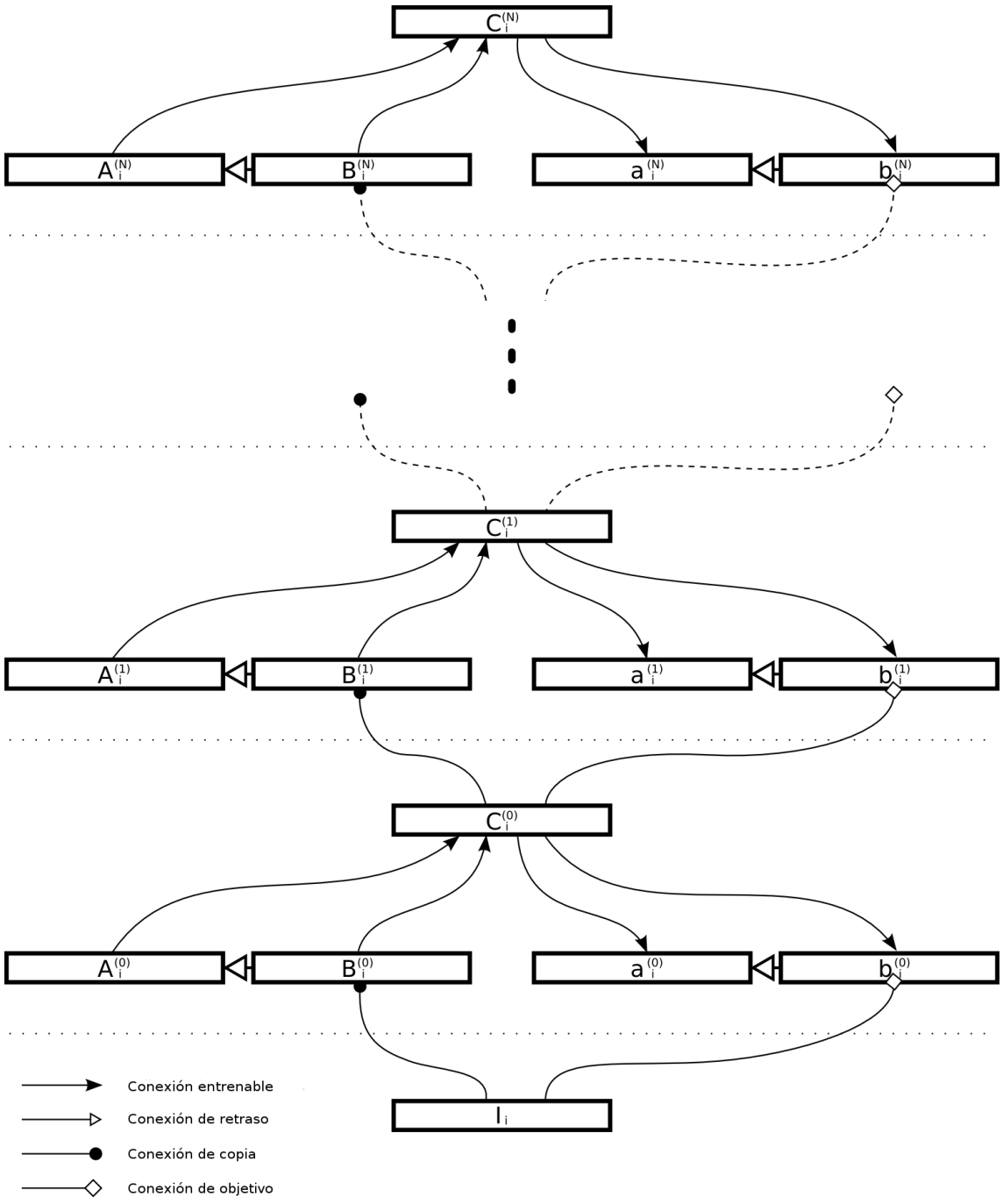

Figura 2.3.: Diagrama del modelo.

En cada etapa j la codificación y decodificación correspondiente a la posición i dentro de la secuencia está dada por:

$$[A_i^{(j)}, B_i^{(j)}] \rightarrow [C_i^{(j)}] \rightarrow [a_i^{(j)}, b_i^{(j)}]$$

A su vez cada etapa j producirá una secuencia $C^{(j)} = [C_0^{(j)}, C_1^{(j)}, ...C_M^{(j)}]$ en donde cada $C_i^{(j)}$ es también un vector [47]. Uno de los objetivos de esta arquitectura es que cada etapa produzca una secuencia de longitud menor o igual durante la codificación. Es decir, secuencias que cumplan $|C^{(j+1)}| \leq |C^{(j)}|$ hasta que finalmente $|C^{(N)}| = 1$, en donde N es la cantidad de etapas.

Las conexiones entrenables, de copia, con retraso y de patrón objetivo están indicadas con distintos tipos de lineas en la misma figura. Las conexiones de copia y retraso trabajan conjuntamente para definir los pares de patrones utilizando un valor de *paso* que puede ser igual a 1 o 2 y que indica como se avanza sobre la secuencia para la codificación y decodificación. Por ejemplo, para la primera etapa, los patrones de entrada tomando *paso 1* quedarían definidos por:

$$[A_i^{(0)}, B_i^{(0)}] = [I_i, I_{i+1}]$$

El valor de *paso*, que es propio de cada etapa, permite definir como se construirá la secuencia de entrada de un nivel a partir de la secuencia de salida del nivel anterior para la codificación, y viceversa para la decodificación. Con un *paso 1* la cantidad de patrones resultantes es mayor pero se obtiene un mejor nivel de redundancia, con un *paso 2* no hay redundancia pero la longitud de la secuencia se reduce a la mitad. Es posible elegir libremente los valores de *paso* para todas las etapas, pero para obtener un buen balance entre nivel de redundancia y máxima longitud de secuencia lo más conveniente es elegir *paso 1* para las primeras etapas, y *paso 2* en las demás.

Dependiendo del valor de *paso* elegido las equivalencias entre patrones en las secuencias será:

$$\begin{aligned} [A_i^{(j)}, B_i^{(j)}] &= [C_{2i}^{(j-1)}, C_{2i+1}^{(j-1)}] \quad \text{si el paso es } 2 \\ [A_i^{(j)}, B_i^{(j)}] &= [C_i^{(j-1)}, C_{i+1}^{(j-1)}] \quad \text{si el paso es } 1 \end{aligned}$$

Notar que esto hace que $|C^{(j)}| = 2|C^{(j+1)}|$ si *paso 2* y $|C^{(j)}| = |C^{(j+1)}| + 1$ si *paso 1*, cumpliendo con la condición $|C^{(j+1)}| < |C^{(j)}|$ requerida. Además notar que en los casos en donde el *paso* es 1 se cumple $A_{i+1}^{(j)} = B_i^{(j)}$, esto introduce el nivel de redundancia en los patrones al momento de la decodificación que hace al modelo más robusto [69, 123]. En cada caso, los patrones objetivo para $[a_i^{(j)}, b_i^{(j)}]$ son iguales a los valores de $[A_i^{(j)}, B_i^{(j)}]$ previamente determinados.

Si en alguna etapa de *paso 2* la longitud de la secuencia es impar se utilizará un patrón terminal T para completar el par de patrones necesarios. Este patrón terminal puede estar formado por un vector de ceros o unos de la dimensión adecuada según convenga.

$$[A_i^{(j)}, B_i^{(j)}] = [C_{2i}^{(j-1)}, T]$$

Finalmente en la última etapa se deberá obtener la secuencia $C^{(N)}$ de longitud 1, es decir, conformada por un solo patrón $C_0^{(N)}$.

La decodificación de un patrón $C_0^{(N)}$ en una secuencia se realiza siguiendo un procedimiento similar. La red correspondiente a la etapa j produce pares de patrones a partir de una secuencia $C^{(j)}$.

$$C_i^{(j)} \rightarrow [a_i^{(j)}, b_i^{(j)}]$$

A partir de la secuencia de pares de patrones se genera una secuencia $C^{(j-1)}$ según.

$$\begin{aligned} C^{(j-1)} &= [\ a_0^{(j)},\ b_0^{(j)},\ a_1^{(j)},\ \ldots\ a_M^{(j)},\ b_M^{(j)}] \ \text{ si el paso es 2} \\ C^{(j-1)} &= [\ a_0^{(j)},\ \tfrac{b_0^{(j)}+a_1^{(j)}}{2},\ \tfrac{b_1^{(j)}+a_2^{(j)}}{2},\ \ldots\ \tfrac{b_{M-1}^{(j)}+a_M^{(j)}}{2},\ b_M^{(j)}] \ \text{ si el paso es 1} \end{aligned}$$

Hasta llegar a la primera etapa en donde se genera la secuencia de salida siguiendo las mismas reglas. Finalmente cada patrón puede ser traducido a un símbolo siguiendo algún criterio de similitud.

2.4. Detalles de Implementación y Uso

Una de las ventajas de la organización en etapas es que, a pesar de tratarse de una arquitectura profunda, el entrenamiento puede realizarse progresivamente. Es decir que cada etapa puede ser entrenada independientemente en forma sucesiva. Esto no solo permite dividir el proceso de aprendizaje sino que también da la posibilidad de pre-entrenar las primeras etapas para que después puedan ser utilizadas en distintos modelos.

Adicionalmente, como cada etapa es una red que toma pares de patrones, se pueden separar todos los pares de patrones distintos que ocurren en el conjunto de secuencias y usarlos directamente para el entrenamiento. Esto, en combinación con la naturaleza progresiva de las etapas, hace que no sea necesario definir la arquitectura completa del modelo por anticipado, ya que la cantidad de repeticiones de cada patrón puede servir para estimar en que medida será posible reducir la dimensión para la siguiente etapa. De todos modos la única forma segura de determinar las dimensiones óptimas es experimentalmente, pero en general se puede contar con una reducción de la dimensión de entre $^2/_3$ y $^3/_4$ con respecto al par de patrones combinados en la etapa anterior.

En la Figura 2.4 se puede apreciar como se combinan los pares de patrones en un modelo con 8 etapas, con *paso 1* en las primeras 3 etapas, y *paso 2* en las 5 últimas. En la parte inferior se representa una secuencia de entrada compuesta por 35 símbolos, la máxima longitud de secuencia válida para esta arquitectura. Para calcular esta longitud

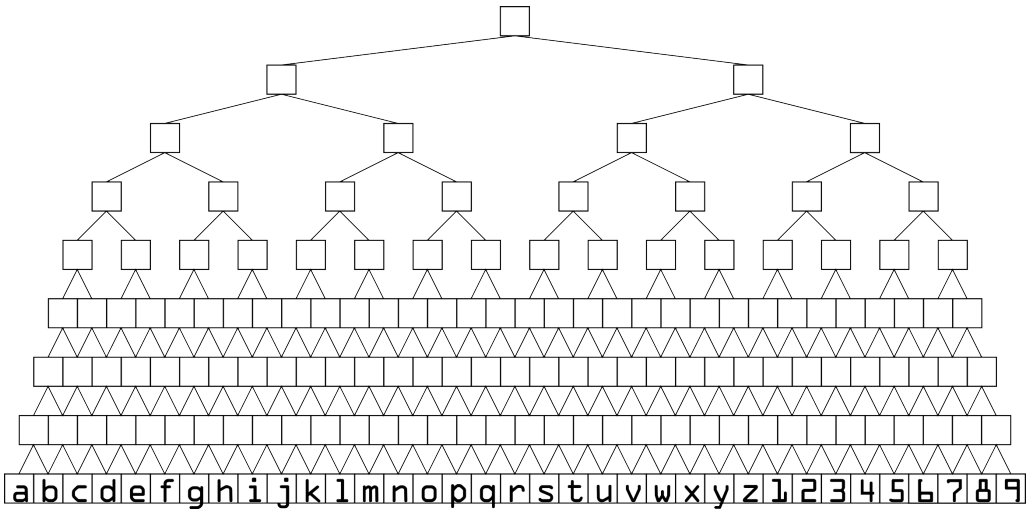

Figura 2.4.: Esquema de codificación de una secuencia repitiendo las conexiones de cada etapa según el paso.

máxima se puede comenzar por la etapa superior, en donde la representación consiste en una secuencia de uno, e ir duplicando esta longitud para las etapas de *paso 2*, o incrementándola en uno para las etapas de *paso 1*, a medida que se va descendiendo hasta llegar a la etapa de entrada. Notar que en este esquema las capas del auto-codificador fueron repetidas en cada etapa por cada par de patrones que debe codificar, en verdad cada nivel solo consiste en dos capas de entrada, dos capas objetivo, y una capa oculta.

2.5. Conclusiones

La arquitectura propuesta es suficientemente adaptable como para permitir la codificación de secuencias de distintos tipos siempre que los símbolos o elementos que las componen puedan ser representados como patrones vectoriales. La principal restricción es la limitación en la longitud máxima de secuencia, que podría ser un inconveniente para varios dominios de datos. Pero considerando su aplicación al lenguaje natural es razonable pensar en arquitecturas que puedan manejar este tipo de datos sin mayores problemas, ya que para los lenguajes occidentales la longitud en letras de las palabras, y en palabras de las sentencias es relativamente corta.

La organización en múltiples etapas no sólo presenta la ventaja de la posibilidad de un entrenamiento progresivo, sino que también permite capturar información a distintos niveles de regularidad. Esto no solo le brinda al modelo suficiente memoria sin sacrificar la capacidad de generalización, sino que con las etapas de *paso 1* se introduce un nivel de redundancia que lo hace más robusto ante errores.

Adicionalmente en cada etapa se codifican pares de patrones que representan dos sub-

secuencias de la secuencia original de modo que el final de una subsecuencia coincide con el comienzo de la otra. Esto hace que no solo se aprendan las regularidades de los patrones, correspondiente a la superposición de las secuencias, sino también en que contexto ocurren, es decir las partes no superpuestas al comienzo y final.

Los resultados producidos por algunos modelos actuales, como los basados en mecanismos de atención, pueden ser superiores, pero su complejidad y costo computacional los hacen menos accesibles para el entrenamiento. El modelo propuesto se presenta como una alternativa basada en una técnica sencilla y escalable para el procesamiento de secuencias en general.

Parte de los contenidos y resultados presentados en este capítulo fueron previamente publicados en el artículo "Effective vector representations for variable length symbol sequences" [41] como parte de Computer Science & Information Technology Conference Proceedings vol. 7, no. 6, p. 27-34. (2017)

3. Codificación Dispersa y Codificación Distribuida

3.1. Introducción

Según la Teoría de Prototipos los significados no pertenecen a categorías discretas sino que se relacionan con distintas características[1] en forma graduada de acuerdo a su similitud con prototipos [35, 63]. La representación de características utilizando vectores de valores continuos en intervalos acotados es una práctica común en varias estrategias de aprendizaje automático[46]. Este tipo de vectores, en los que cada dimensión puede ser interpretada como un grado de similitud con una característica, pueden servir como buena representación del conocimiento.

Una de las propiedades de los autocodificadores es la de producir un tipo codificación similar a los vectores de características [123]. El tipo de respuesta producida normalmente por estos se la conoce como codificación distribuida, ya que los niveles de activación se distribuyen de forma más o menos uniforme entre todas las unidades [78, 116]. La codificación distribuida no es la única forma de representar vectores de características, y en muchos casos ni siquiera es la mejor. Alternativamente también existe la llamada codificación local, en la que generalmente una sola unidad se activa por vez [57], y la codificación dispersa (sparce coding), que en cierta forma funciona como una variante intermedia entre la codificación local y la distribuida, ya que se caracteriza por un mayor nivel de activación para solo una porción de las unidades a la vez, mientras que el nivel de activación de las demás tiende a cero. En muchos casos la codificación dispersa es considerada como la mejor para representar vectores de características [104]. En este capítulo se exploran las posibilidades de varias estrategias de codificación para su aplicación en la representación del conocimiento.

3.2. Antecedentes y Motivación

La codificación local es utilizada generalmente en problemas de clasificación, la unidad activa indica a que categoría pertenece la entrada o da una probabilidad de pertenencia a cada clase. Por ejemplo, existe una función de activación conocida como *winner-take-all* [57, 99] en la que una sola unidad responde con valor de 1 mientras que las demás

[1] en inglés "features".

permanecen sin activarse con un valor en 0. Alternativamente, la función conocida como *softmax* responde con una distribución de probabilidades, en donde se espera que la unidad que representa la clase a la que pertenece la entrada sea la que tenga un mayor nivel de activación [9].

Existen varias formas de producir codificación dispersa y, aún, distintos tipos de codificaciones posibles. Por ejemplo, un método generalmente asociado a la codificación dispersa es el llamado Análisis de las Componentes Independientes (ICA). Este método trata de encontrar una forma de combinación lineal que explique los datos a partir de variables estadísticamente independientes. El resultado es un tipo codificación dispersa, pero de naturaleza lineal que, generalmente, tiende a apartarse de lo que es considerado una buena representación [56, 18, 52].

Otra forma comúnmente utilizada consiste en la inclusión de un término de penalización en la función de costo, por ejemplo una norma L_1 sobre los valores de activación de la salida. Esto tiene la ventaja de poder ponderar el nivel de dispersión mediante un parámetro λ, pero dificulta el entrenamiento debido a que no es posible derivar una regla de aprendizaje directamente de esta nueva función de costo, forzando el cálculo de las correcciones alternativamente entre los dos términos, lo cual tampoco garantiza una buena codificación durante el testeo [27, 39, 68].

Por otra parte, un método relativamente reciente y sencillo son las llamadas Unidades Lineales Rectificadas (ReLU). En este caso la activación es determinada mediante la función rectificada $f(x) = \max(0, x)$, lo cual lo hace muy práctico en su implementación. Sin embargo, la naturaleza no acotada de esta función, en combinación con métodos de entrenamiento como el de retro-propagación del error, puede conducir a configuraciones inestables, como un crecimiento desproporcionado de los pesos [16].

Finalmente la codificación distribuida es quizás la más común, ya que es la que se obtiene, por ejemplo, en las capas ocultas de los perceptrones multi-capa [46, 108]. Como ya se mencionó antes, los autocodificadores producen este tipo de representación, precisamente porque está formados por capas de perceptrones y su respuesta es el resultado de la activación en su capa oculta. Sin embargo, existen también algunas variantes del autocodificador simple como el llamado *tied*, en donde los pesos entre el *encoder* y el *decoder* son compartidos, o el *denoise*, en donde se agrega ruido en la entrada que debe ser corregido, que producen distintos tipos de codificaciones utilizando esencialmente la misma arquitectura.

Por otra parte, hay algunos modelos cuyo comportamiento puede ser considerado como equivalente al de un autocodificador bajo ciertas condiciones, pero que producen codificaciones distribuidas mediante sus propias estrategias. Por ejemplo, el modelo conocido Máquina Restringida de Boltzmann Continua (CRBM), que produce una activación basada en las probabilidades de una salida dada una entrada ponderada, pero que es más difícil de entrenar [111, 83]. O una variante del modelo de Oja conocida como *Nonlinear PCA*, que utiliza una sencilla regla de aprendizaje en combinación con funciones de activación sigmoideas [84].

Cada una de estas estrategias presenta propiedades interesantes pero con distintas limitaciones, especialmente al tratar de aplicarlas para obtener una codificación que sea adecuada a las necesidades planteadas para la representación del conocimiento. Quizás una alternativa más viable sea buscar un estrategia propia para generar la codificación deseada, tratando de capturar algunas de las ventajas expuestas, y evitando los principales inconvenientes planteados.

3.3. Representación Dispersa Regulada

Teniendo en cuenta que la codificación dispersa puede ser vista como una solución intermedia entre una codificación local y una codificación distribuida, se plantea la posibilidad de construir un tipo de estrategia de codificación dispersa cuya proporción de activación pueda ser elegida para que tienda a comportarse con cualquier nivel de dispersión entre ambos extremos.

Un tipo de función de activación comúnmente usada que tiende a generar una codificación local es la conocida como *softmax* (Ecuación 3.1), y que en estadística puede ser utilizada para representar una distribución de categorías, es decir, la distribución de probabilidades sobre N categorías posibles. Este tipo de función también es conocida como regresión logística multinomial, ya que es una generalización del caso de regresión logística para $N = 2$. A su vez, la función *logística* (Ecuación 3.2) suele ser utilizada como función de activación para generar codificaciones distribuidas, por ejemplo en las capas ocultas en los perceptrones multi-capa.

$$\Pr(Y_j = k) = \frac{e^{\beta_k \cdot x_j}}{\sum_h^N e^{\beta_h \cdot x_j}} \tag{3.1}$$

$$\Pr(Y_j = 0) = \frac{e^{\beta_0 \cdot x_j}}{e^{\beta_0 \cdot x_j} + e^{\beta_1 \cdot x_j}} = \frac{e^{-\beta \cdot x_j}}{1 + e^{-\beta \cdot x_j}} \qquad \beta = -(\beta_0 - \beta_1)$$

$$\Pr(Y_j = 1) = 1 - \Pr(Y_j = 0) = \frac{1}{1 + e^{-\beta \cdot x_j}} \tag{3.2}$$

Esto sirve como referencia a la relación que existe entre los dos tipos de codificaciones en los extremos, aunque una función de distribución de probabilidades no sea exactamente el tipo de solución que se está buscando. Preferentemente se debería contar con un parámetro que permita moverse entre ambos comportamientos.

Para esto consideraremos inicialmente el problema de una arquitectura de red consistente en una capa de entrada X de dimensión M y una capa de salida Y de dimensión N, en

donde preferentemente será $M > N$. En el vector X se puede incluir una unidad extra fijada en 1 que sirva como umbral. Se espera que los valores de activación en Y estén acotados en el intervalo $[0, 1]$. Las conexiones entre ambas capas estarán indicadas por la matriz W que será inicializada al azar.

El estímulo que recibe cada unidad Y_j de la capa de salida puede notarse como:

$$z_j = e^{\sum_i^M X_i W_{ij}} \tag{3.3}$$

Tomar el exponencial del producto interno hace que las unidades cuyos pesos se parezcan, al menos en parte, a ciertas características presentes en la entrada reciban un mayor estímulo.

Para controlar qué porción de las unidades tendrán una mejor oportunidad de activarse se utilizará el parámetro k. En donde, por ejemplo, para un valor de $k = 1$ se esperaría que el comportamiento fuera similar al de la función *softmax*, con una sola unidad con un mayor nivel de activación que las demás, o con un valor de $k = {}^{N}/_{2}$ similar al de la *logística*, con todas las unidades con aproximadamente las mismas posibilidades de activarse. Sin embargo, este parámetro no indica directamente cuantas unidades estarán activas por vez, y ni siquiera tiene por que limitarse a valores enteros.

La función de transferencia utilizada será una logística desplaza en ${}^{1}/_{2}$ y con una pendiente β relativamente pronunciada, por ejemplo con valores entre 5 y 10, para que sature rápidamente al tender a los valores 0 y 1.

$$s(x) = \frac{1}{1 + e^{-\beta(x - \frac{1}{2})}} \tag{3.4}$$

Para poder estimar el nivel de activación de las unidades se debe comenzar por calcular el estímulo promedio que reciben.

$$\tilde{z} = \frac{\sum_h^N z_h}{N} \tag{3.5}$$

Usando esto es posible estimar un máximo nivel de activación $(2\tilde{z})$ y a partir de ahí determinar un punto de corte p que esté k pasos por debajo de este máximo estimado $(k \cdot {}^{2\tilde{z}}/_{N})$. Es decir, las unidades con un nivel de estimulo superior a p serán las que tiendan a activarse.

$$\begin{aligned} p &= 2\tilde{z} - k\frac{2\tilde{z}}{N} \\ &= 2\tilde{z}(1 - \frac{k}{N}) \end{aligned} \tag{3.6}$$

Finalmente se busca que este valor coincida con el punto de inflexión de la función de transferencia, de modo que tienda a reducir los estímulos menores a éste y a aumentar los mayores. Esto puede lograrse multiplicando los valores de estímulo por un factor de correcciónc de modo que $c \cdot p = 1/2$.

$$c \cdot 2\tilde{z}(1 - \frac{k}{N}) = \frac{1}{2}$$

$$c = \frac{1}{\tilde{z}} \cdot \frac{1}{4 \cdot (1 - k/N)} \tag{3.7}$$

$$c = \frac{1}{\sum z_h} \cdot \frac{N^2}{4 \cdot (N - k)}$$

Con lo cual el nivel de activación final para la unidad j queda dado por:

$$Y_j = s\left(\frac{z_j}{\sum_h^N z_h} \cdot \frac{N^2}{4(N-k)}\right) \tag{3.8}$$

En la figura Figura 3.1 se puede apreciar el estímulo que reciben 16 unidades junto con el valor estimado de p para $k = 4$, y el efecto que éste tiene sobre los niveles finales de activación. Las unidades más activas son las que reciben una mayor corrección de modo que la próxima vez que un patrón similar aparezca en la entrada las diferencias en los niveles de activación serán aún más marcados.

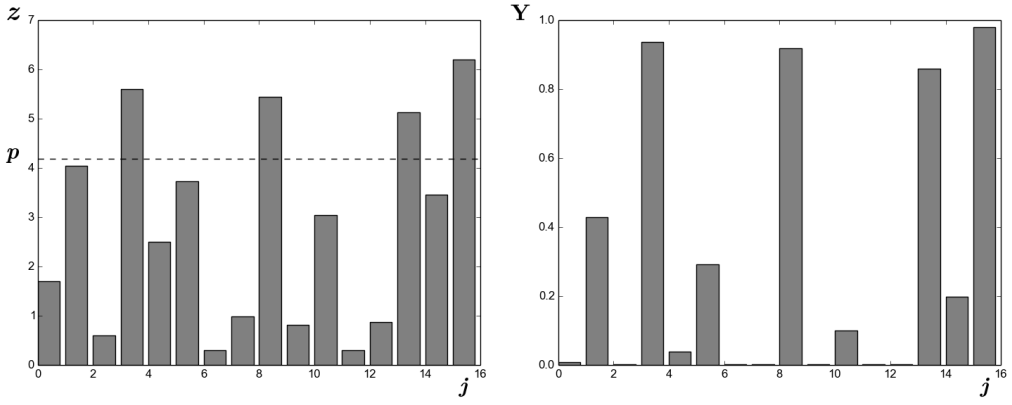

Figura 3.1.: Niveles de estimulo y activación para 16 unidades con $k = 4$.

3.4. Derivación de la Regla de Aprendizaje

Para realizar el aprendizaje es posible calcular las correcciones en forma análoga a otros autocodificadores, es decir, a partir de en una estimación de cuánto se puede describir de la entrada X a partir de las unidades activas en la salida. La forma más sencilla consiste en una aproximación lineal $\tilde{X}$ de la entrada calculada a partir de las unidades Y_j de la capa de salida y los pesos W.

$$\tilde{X}_i(Y, W) = \sum_j^N Y_j W_{ij} \tag{3.9}$$

El gradiente de las correcciones sobre los pesos se determina derivando una regla de aprendizaje a partir de una función de costo E que minimice la suma de los errores cuadráticos a partir de la diferencia entre la entrada original y la aproximación calculada. Al igual que en la expresión de $\tilde{X}$ se incluyeron los términos Y y W explícitamente para especificar su origen, también se incluirán explícitamente X y W en la expresión de Y. Además, como el costo se calcula sobre todo el conjunto de datos, se indica con el superíndice u la instancia correspondiente, y con un subíndice i la variable correspondiente.

$$E(X^U, W) = \frac{1}{2} \sum_u \sum_i \left(X_i^u - \tilde{X}_i(Y(X^u, W), W) \right)^2 \tag{3.10}$$

$$E_i^u(X^u, W) = \frac{1}{2} \left(X_i^u - \tilde{X}_i(Y(X^u, W), W \right)^2 \tag{3.11}$$

$$\frac{\partial E_i^u(X^u, W)}{\partial W_{i,j}} = \frac{\partial E_i^u(X^u, W)}{\partial \tilde{X}_i(Y(X^u, W), W)} \cdot \frac{\partial \tilde{X}_i(Y(X^u, W), W)}{\partial W_{i,j}} \tag{3.12}$$

$$\frac{\partial E_i^u(X^u, W)}{\partial \tilde{X}_i(Y(X^u, W), W)} = \frac{1}{2} \cdot 2 \left(X_i^u - \tilde{X}_i(Y(X^u, W), W) \right) \tag{3.13}$$

$$\frac{\partial \tilde{X}_i(Y_j(X_i^u, W), W)}{\partial W_{i,j}} = Y_j(X^u, W) \tag{3.14}$$

$$\frac{\partial E_i^u(X^u, W)}{\partial W_{i,j}} = (X_i^u - \tilde{X}_i) \cdot Y_j \tag{3.15}$$

La regla de aprendizaje resultante (Ecuación 3.15) de la derivación de la función de costo es idéntica a la regla de aprendizaje Hebbiano planteada originalmente por Oja [85]. Esta regla, utilizada con unidades de activación lineal, es capaz de aprender una transformación en un espacio equivalente al encontrado a partir de un Análisis de las Componentes Principales (PCA). Es decir que, aún cambiando la forma en que se determina la activación en la capa de salida, las correcciones en los pesos se realizan de la misma manera. Esto es importante para poder asegurar el uso de esta técnica con estrategias de entrenamiento basadas en el descenso por gradiente.

Otra solución similar, el modelo mencionado anteriormente también propuesto por Oja llamado *Nonlinear PCA* [84], utiliza la misma regla de aprendizaje pero con la activación determinada por una función logística, y cuyo comportamiento produce un tipo de codificación dispersa asociada a la separación de variables independientes (ICA) [53, 56, 55]. Por sus características este tipo de modelo es similar a la variante de autocodificador llamada *Tied Autoencoder*, que comparte sus pesos entre la capa oculta y de salida.

Esto es de especial interés dado que se estima que para que una codificación dispersa sea eficaz solo se deben activar entre un 12 % y un 25 % de las unidades a la vez. Para lograr esto se suele recurrir a lo que se llama una codificación supra-completa (over-complete), es decir, el caso en donde $N > M$ [86]. Sin embargo, en el caso de un autocodificador típico, se busca obtener una representación en un espacio de dimensión menor, es decir una codificación sub-completa (under-complete) [93]. En las siguiente pruebas se utilizaron distintas arquitecturas variando el factor de dispersión y entrenadas para tratar de capturar las características que mejor representen a los datos de entrada.

Método	**N=48 p=12.5 %**	**N=64 p=12.5 %**	**N=72 p=12.5 %**
ReLU+λL_1	11.118	10.217	9.981
SparseTopK	29.754	19.698	16.935
método propuesto	6.903	4.564	4.012

	N=48 p=25 %	**N=64 p=25 %**	**N=72 p=25 %**
ReLU+λL_1	10.787	9.833	9.210
SparseTopK	9.936	7.152	6.161
método propuesto	5.336	3.987	3.831

	N=48 p=50 %	**N=64 p=50 %**	**N=72 p=50 %**
ReLU+λL_1	9.253	8.580	8.114
SparseTopK	4.346	3.426	3.101
método propuesto	4.478	3.721	3.387

Cuadro 3.1.: Comparación del error de entrenamiento (MSE) entre los distintos métodos, variando la cantidad de unidades de salida (N), y la dispersión esperada (p).

Método	N=48 p=12.5 %	N=64 p=12.5 %	N=72 p=12.5 %
ReLU+λL_1	0.00136	0.00131	0.00123
SparseTopK	0.00804	0.00516	0.00458
método propuesto	0.00181	0.00122	0.00111

	N=48 p=25 %	N=64 p=25 %	N=72 p=25 %
ReLU+λL_1	0.00138	0.00123	0.00120
SparseTopK	0.00263	0.00188	0.00162
método propuesto	0.00144	0.00106	0.00105

	N=48 p=50 %	N=64 p=50 %	N=72 p=50 %
ReLU+λL_1	0.00123	0.00114	0.00107
SparseTopK	0.00117	0.00092	0.00083
método propuesto	0.00121	0.00102	0.00091

Cuadro 3.2.: Comparación del error de evaluación (MSE) entre distintos modelos entrenados, variando la cantidad de unidades de salida (N), y la dispersión esperada (p).

3.5. Experimentación y Resultados

Para poder demostrar la efectividad de la técnica propuesta se realizaron varias pruebas comparando los resultados obtenidos con los de dos de las técnicas más comúnmente utilizadas para producir codificaciones dispersas. A saber, unidades lineares rectificadas con un término de penalización en la función de costo (ReLU+λL_1) y la selección de los mayores k valores de activación (SparseTopK).

Todos los modelos fueron entrenados por el mismo número fijo de épocas. Los resultados que se muestran fueron obtenidos promediando la respuesta de varias corridas independientes. Las pruebas para todos los métodos fueron hechas variando la cantidad de unidades de salida (N), que corresponde al número de características, y la proporción de unidades activas simultáneas esperadas (p). El conjunto de datos utilizados fue construido a partir fragmentos de 6×6 pixels, con valores acotados entre 0 y 1, provenientes del dataset MNIST. Los resultados analizados consistieron en: la velocidad de convergencia, estimada a partir del error de entrenamiento, el error sobre datos de testeo, a partir de Error Cuadrático Medio (MSE, del inglés Mean Squared Error), y la dispersión en la codificación producida, obtenida a partir de aplicar la norma L_1 a la activación de salida.

En la Tabla 3.1 se muestra el valor de costo o pérdida de entrenamiento calculado a partir del error cuadrático medio (MSE) después de 20 épocas para un conjunto de datos consistente en 60 mil instancias. Esta medida puede servir como una estimación de la velocidad de convergencia a una buena solución de los distintos métodos bajo diferentes condiciones. Para los casos de una mayor dispersión el método SparseTopK

Método	N=48 k=6	N=64 k=8	N=72 k=9
ReLU+λL_1	6.845	6.130	6.142
SparseTopK	14.347	16.619	17.290
método propuesto	11.516	14.764	16.605

	N=48 k=12	N=64 k=16	N=72 k=18
ReLU+λL_1	7.510	6.777	6.459
SparseTopK	19.928	24.028	26.045
método propuesto	14.256	18.618	21.705

	N=48 k=24	N=64 k=32	N=72 k=36
ReLU+λL_1	9.593	8.802	8.142
SparseTopK	39.559	45.221	45.808
método propuesto	23.907	32.037	35.882

Cuadro 3.3.: Comparación de la dispersión en la codificación, medida como la norma L_1, variando la cantidad de unidades de salida (N) y la cantidad de unidades activas esperada (k).

parece converger marginalmente más rápido, pero el método propuesto ofrece buenos resultados consistentemente en todas las situaciones.

En la Tabla 3.2 se comparan los valores de error de testeo, también utilizando MSE, para modelos completamente entrenados, sobre un conjunto de datos de 10 mil instancias nunca vistas durante el entrenamiento. Esta medida es las que se suele utilizar para comparar la eficacia de distintos métodos, sin embargo como se puede ver, salvo por pequeñas diferencias, el rendimiento es similar en todos los casos.

Finalmente, en la Tabla 3.3 se trata de demostrar el nivel de dispersión obtenido en la codificación. Como en los casos anteriores, los datos consisten en parches de 6×6 provenientes de MNIST divididos en 60 mil para entrenamiento y 10 mil para testeo. Para este fin se utiliza como medida la norma L_1 tomada sobre el nivel de activación de las unidades de salida. Para que esta medida pueda ser interpretada más fácilmente, en lugar de indicar la proporción de unidades activas (p) como en los casos anteriores, se indica la cantidad de unidades que se esperan activas (k), aunque ambos valores representan lo mismo. En este caso, bajo condiciones con mayores restricciones las unidades ReLU parecen ofrecer los mejores resultados pero, como en las pruebas anteriores, el método propuesto es consistentemente mejor en una mayor variedad de casos.

3.6. Conclusiones

En los esquemas de reducción de dimensión la codificación distribuida, comúnmente producida por autocodificadores simples, está entre las más aceptadas. Sin embargo

cuando las variables de entrada no tienen la misma distribución entre distintas instancias, por ejemplo cuando los valores tienden a agruparse en clusters, un tipo de codificación dispersa es más efectiva para captar las características que mejor representan a los datos.

Técnicas típicamente utilizadas para producir codificaciones dispersas, como las unidades lineales rectificadas con un término de penalización o conservando sólo los mayores valores de activación, suelen presentar una mayor complejidad para el entrenamiento y en su implementación.

El método aquí propuesto ofrece mejores resultados en una mayor variedad de escenarios, y puede ser implementado de forma muy sencilla como un tipo de función de activación y entrenado por una estrategia de descenso por gradiente. Esto le permite funcionar como una variante de autocodificador que pueda ser integrada fácilmente en una arquitectura profunda.

Parte de los contenidos y resultados presentados en este capítulo fueron previamente publicados en el artículo "Coding with Logistic Softmax Sparse Units" como parte de AIFU 2020, 6th International Conference on Artificial Intelligence and Applications. (2020)

4. Agilizando el Descenso por Gradiente

4.1. Introducción

Cuando se trata de minimizar una función de costo altamente no-convexa utilizando alguna estrategia de descenso por gradiente existen varios problemas potenciales. Si se considera una sola dirección dentro del espacio de búsqueda pueden aparecer los mínimos locales [38], en donde se debe primero ascender para después poder alcanzar un mínimo menor/global, o las mesetas (plateaus) [118], en donde la derivada parcial tiene un valor muy bajo aunque se encuentre en un punto alejado de un mínimo. Por supuesto que en un espacio multidimensional mayor estos problemas pueden combinarse para dar lugar a otros como los mínimos parciales (ravines) [15], en donde existe un mínimo local en una dirección del subespacio y un plateau en otra, y los puntos de ensilladura (saddle) [23], en donde las derivadas parciales son bajas porque se forma un mínimo parcial en una dirección del subespacio y un máximo en otra. Aún considerando una sola dirección de búsqueda puede verse fácilmente como un algoritmo que utilice la información del gradiente para tratar de descender a un buen mínimo puede encontrarse en problemas.

El algoritmo de Descenso de Gradiente Omisivo (Gradient Omissive Descent) aquí propuesto se presenta como una potencial solución para estos problemas y una alternativa más efectiva que los métodos comúnmente utilizados para el entrenamiento supervisado de redes neuronales artificiales multicapa [40].

4.2. Antecedentes y Motivación

Estrategias como Stochastic Gradient Descent (SGD) [108] y Simulated Annealing (SA) [102, 59] introducen un elemento aleatorio en la búsqueda del mínimo que permite superar hasta cierto punto el problema de los mínimos locales, mientras que otros como Gradiend Descent with Momentum (GDM) [82] o el uso de parámetros adaptativos [22] permiten acelerar la búsqueda en el caso de los plateaus. En combinación con métodos de entrenamiento incremental o mini-batch estas técnicas pueden ser efectivas hasta cierto grado aún en las situaciones de los ravines y saddle points, pero utilizan hiper-parámetros globales que deben ser elegidos para cada problema.

Otras técnicas como Conjugate Gradient Descent (CGD) [81], u otras que incorporan información sobre las derivadas segundas, como el Método de Newton y el Algoritmo de

Levenberg-Marquardt [126] permiten acelerar la búsqueda mediante un mejor análisis de la función de costo, pero en general tienen un alto costo computacional y un nivel de complejidad mayor que dificulta su implementación y uso.

Finalmente algoritmos más recientes como Quick Back-Propagation (QBP) [126], Resilient Back-Propagation (RBP) [101], y Adaptative Momentum Estimation (AME) [133, 28] utilizan parámetros adaptativos independientes para cada dirección de búsqueda, por lo cual no requieren la elección de hiper-parámetros globales, y pueden adaptar su estrategia para lidiar con distintos tipos de problemas. Sin embargo, aún en estas condiciones, pueden quedar atrapados en malas soluciones (poor minima) sin forma de escapar. Además todas estas técnicas requieren de entrenamiento batch, lo cual en muchas situaciones no solo puede ser inconveniente sino imposible o altamente impráctico.

La mayor dificultad en la búsqueda de un buen mínimo (global o no) reside en que estos algoritmos tratan de tomar buenas decisiones basándose únicamente en información local y valores que, en muchos casos, son el resultado de estimaciones de estimaciones. Es decir, con información incompleta e imprecisa es difícil tomar decisiones inteligentes.

Quizás una mejor solución sería únicamente tratar de que al final de cada época la probabilidad de que el costo haya descendido sea mayor que la probabilidad de que haya aumentado, y en caso de que haya aumentado que tenga una oportunidad razonablemente buena de escapar a otra posición dentro de la función de costo. De esta manera, en lugar de tratar de asegurar la convergencia mediante el análisis de la función de costo se trata de que estadísticamente tienda a un buen mínimo.

4.3. El Algoritmo Propuesto

El método aquí presentado, en cierta forma, toma varias de sus ideas de otros algoritmos. Por ejemplo, de SGD la introducción de un elemento aleatorio en el descenso, de CGD la exploración de la función de costo, aunque no lineal en este caso, y de RBP el uso del signo del gradiente como indicador de la dirección de búsqueda, *omitiendo* su magnitud [126].

La idea principal es que durante el entrenamiento se avance sobre la función de costo con un paso aleatorio dentro de un radio de búsqueda, de forma que estadísticamente tienda al mínimo y que este radio se vaya reduciendo al ir aproximándose a la solución.

En cada época se divide el conjunto de datos de entrenamiento en mini-lotes. Por cada uno de estos mini-lotes se hace una estimación parcial de la función de costo E, y se calcula el gradiente mediante retro-propagación del error estándar. Después de esto se modifican los pesos usando una magnitud aleatoria acotada por un valor R, pero utilizando la dirección (signo) de los $\triangle W$ ya calculados. Al final de cada época se evalúa si el costo disminuyó con respecto a pasos anteriores y, si corresponde, se realizan los ajustes necesarios en R.

Si consideramos cada mini-lote como una muestra del conjunto de datos puede pensarse que en cada época se hace una estimación Montecarlo de la función de costo. De forma

similar, la reducción del radio de búsqueda puede verse como una especie de reducción de la temperatura en simulated annealing [75]. Sin embargo, hacer que R descienda en forma predeterminada en función de la cantidad de épocas o del costo es impráctico debido a la variabilidad entre problemas. Por esto se prefirió utilizar una estrategia en donde los ajustes a R se realicen dinámicamente dependiendo del costo relativo [71]. Esto lo hace más robusto pero los ajustes deben ser sutiles, del orden del 1 %, debido a que, por la misma naturaleza en la estimación de E, podrían producirse oscilaciones.

La regla más elemental para ajustar R podría exponerse entonces como: si el costo aumentó con respecto a la última época, reducir el radio de búsqueda. Esto es relativamente efectivo en una variedad de escenarios, pero es aún posible mejorar estos resultados reemplazando algunos parámetros globales por locales [132]. Por ejemplo, al final de cada época podemos tener una estimación de la contribución particular de cada unidad al costo total; es decir, el costo por unidad (E_u). De manera análoga, en lugar de utilizar un radio de búsqueda global R, podemos tener un radio de búsqueda individual por unidad (R_u).

Bajo estas condiciones la regla para ajustar los radios de búsqueda podría modificarse como: si el costo total aumentó, reducir el radio de búsqueda para las unidades con mayor error y aumentarlo para las otras. Esta estrategia podría interpretarse pensando en el radio de búsqueda como el nivel de resolución de la función de costo. Si esta no puede seguir disminuyendo se reduce el radio para verla con mayor detalle [22]. Aumentar el radio para las unidades cuyo costo disminuyó ayuda a mantener un equilibrio y, eventualmente, escapar de una mala solución.

Algoritmo 4.1 El algoritmo propuesto.

inicializar W, R, E, t
mientras $(E > min\_error) \wedge (t < max\_epoch)$:
  $E^{[t]} \leftarrow 0$
  para cada lote b:
    $W \leftarrow W + \mathcal{U}(0, R)^{|W|} \times sgn(\frac{\partial E(W,b)}{\partial W})$
    $E^{[t]} \leftarrow E^{[t]} + E(W, b)$
  $dE \leftarrow E^{[t]} - E^{[t-1]}$
  si $dE > 0$:
    para cada unidad u:
      si $dE_u > 0$:
        $R_u \leftarrow R_u \times 0{,}99$
      sino:
        $R_u \leftarrow R_u \times 1{,}01$
  $t \leftarrow t + 1$

4.4. Metodología y Resultados

En todas las pruebas se comparó al método propuesto (GOD) Algoritmo 4.1 con otros tres algoritmos: SGD, GDM, RBP. Estos fueron elegidos para cubrir las alternativas de entrenamiento incremental, en mini-lotes y por lotes correspondientemente. En el futuro también sería posible incluir otros algoritmos como CGD, aunque los resultados podrían no ser fácilmente comparables, y AME, dependiendo de una implementación adecuadamente testeada.

La condición de parada en todas las simulaciones fue dada al superar un número máximo de épocas o al alcanzar un error por debajo de un error mínimo. El término de error utilizado en esta condición de parada fue el dado por el error promedio para todos los patrones de entrenamiento. Por tratarse de datos generados artificialmente el conjunto de entrenamiento pudo mantenerse con un tamaño consistente a lo largo de todas las pruebas.

Como medida de evaluación de la efectividad de cada algoritmo, y por no existir una única condición de parada, se utilizó el valor resultante de calcular ${}^{epoch}/_{max\_epoch} \times {}^{error}/_{min\_error}$, en donde $epoch$ y $error$ corresponden a los valores de finalización, y max_epoch y min_error a los valores en la condición de parada. De este modo, valores menores corresponderán a una mayor efectividad, y en particular, valores menores que 1 corresponderán a los casos en que se haya convergido a una solución.

Para los experimentos se eligieron dos tipos de problemas característicos en la evaluación de esta clase de algoritmos: paridad y auto-codificación. En ambos casos se incluyeron restricciones adicionales con el fin de tener un mayor nivel de control y así poder variar el grado de dificultad [65]. Los resultados finales fueron calculados tomando el promedio de varias simulaciones bajo las mismas restricciones pero con distintos datos.

Paridad:

El problema de paridad, o de XOR para múltiples variables, puede entenderse como el resultado de verificar si la cantidad de unos que aparecen en una lista de variables binarias es par. Para generar los datos en estos experimentos utilizamos variables bipolares ($x_i \in \{-1, 1\}$) en lugar de binarias, con una función par definida como:

$$par(x_0 \cdots x_n) = \begin{cases} 1 & si\ (\sum_{i=0}^{n} u(x_i)) \bmod 2 = 0 \\ -1 & sino \end{cases}$$

En donde $u(x) = 1$ si $x = 1$, e igual a cero en cualquier otro caso.

En su forma más simple este problema consiste en mapear las n entradas a una única salida con la paridad. Lo que lo hace especialmente interesante es que el cambio en cualquier variable de entrada produce un cambio en la salida. Para nuestras pruebas utilizamos 10 variables bipolares $[x_0 \cdots x_9]$ de entrada y 10 variables bipolares $[z_0 \cdots z_9]$ como salida. En donde z_i está definida como $z_i = par(x_0 \cdots x_i)$. Es decir, cada salida

corresponde a la paridad sobre una porción de la variables de entrada. Esto le da un grado de dificultad mayor al problema, ya que la única capa oculta del modelo debe encontrar una representación que permita satisfacer todas las salidas simultáneamente.

Auto-codificación con variables dependientes:

Uno de los factores principales en la efectividad de una auto-codificación es cuan independientes son los datos entre sí. Si los datos están formados por variables completamente independientes y de igual varianza no es posible hacer una reducción de dimensión efectiva [47].

Por esto para las pruebas se generaron conjuntos de datos variando la proporción entre variables independientes y dependientes. Las valores de las variables independientes fueron generados a partir de una distribución aleatoria uniforme acotada entre -1 y 1, y los de las dependientes de una combinación lineal aleatoria de las independientes. Los valores finales son el resultado de aplicar la función signo para poder trabajar en este caso también con datos bipolares.

Sin embargo, en un escenario real, los datos no suelen estar todos correlacionados de la misma manera. Esto también fue simulado dividiendo el conjunto de datos en varias clases. Cada clase no solo utiliza diferentes correlaciones para las variables dependientes, sino que también utiliza distintas posiciones para las variables independientes.

A continuación se presenta una comparación de los resultados obtenidos con lo distintos algoritmos considerados para los problemas antes descriptos. Para cada algoritmo se eligieron los parámetros que ofrecieran mejores resultados. Cada simulación fue corrida por un máximo de 3000 épocas o hasta que el error promedio fuera menor a 10^{-3}. Estos valores fueron elegidos para asegurar que todos los algoritmos tuvieran oportunidad de converger.

Los pesos fueron inicializados independientemente para cada simulación con valores aleatorios extraídos de una distribución uniforme acotada entre $-N^{-\frac{1}{2}}$ y $N^{-\frac{1}{2}}$, en donde N es la cantidad unidades en la capa anterior. Los valores iniciales para R fueron calculados utilizando la expresión $1/N \times M$, en donde N es la cantidad de unidades en la capa anterior y M la cantidad de unidades en la misma capa. Cada mini-lote, tanto para GDM como para GOD, fue definido utilizando un 10 % de las instancias del conjunto de datos elegidos al azar sin repetición.

Paridad:

La arquitectura elegida en este caso contó con una sola capa oculta de 40 unidades. El conjunto de datos estuvo formado por las 2^{10} instancias que es posible generar para los vectores de entrada $[x_0 \cdots x_9]$ y objetivo $[z_0 \cdots z_9]$ que describen el problema según especificaciones anteriores. Para SGD el coeficiente de aprendizaje η utilizado fue de

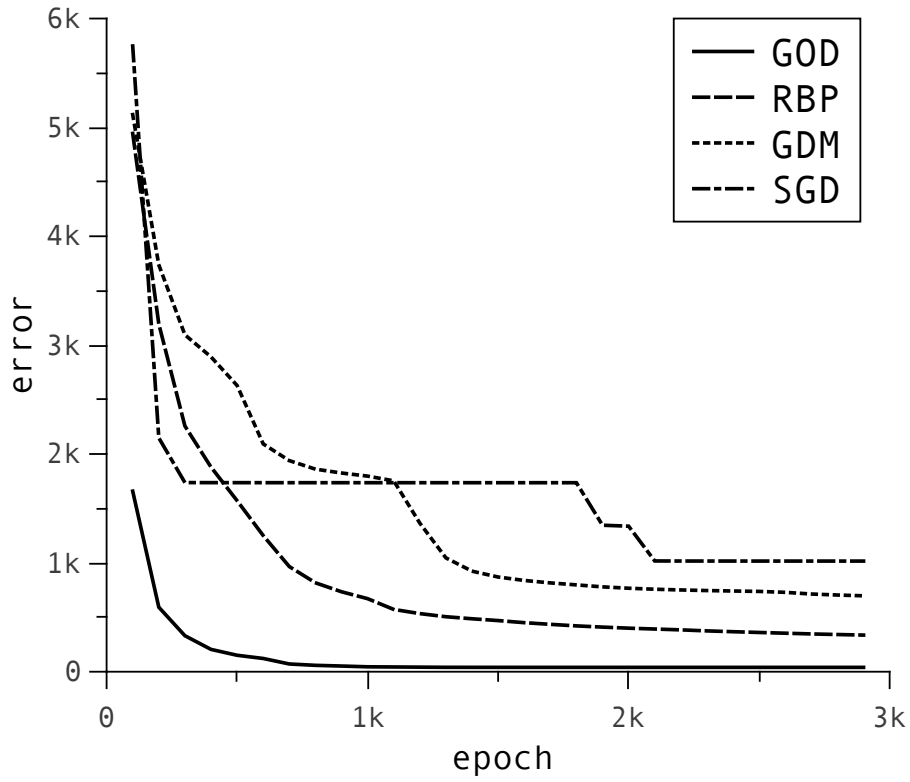

Figura 4.1.: Resultados representativos de un entrenamiento típico para paridad múltiple de varios algoritmos.

$0{,}01$, con GDM $\eta = 0{,}001$ y el factor de momento $\beta = 0{,}5$. Para RBP, en todos los casos, se utilizaron los valores $\eta^+ = 1{,}2$ y $\eta^- = 0{,}5$ propuestos en [101].

En la figura Figura 4.1 se muestra la evolución de la suma de los errores cuadráticos por época para los distintos algoritmos. Estos datos corresponden a una corrida típica de entrenamiento, ya que al promediar varias simulaciones no es posible observar la diferencias en el comportamiento de los distintos métodos. Si bien la naturaleza estocástica propia del entrenamiento introduce grandes variaciones en el camino de descenso para cada algoritmo el orden de convergencia es notablemente consistente. Y en general se puede observar como el costo tiende a disminuir de forma mucha más rápida con el algoritmo propuesto.

Auto-codificación con variables dependientes:

Para las pruebas se utilizó una arquitectura con 100 unidades de entrada y salida, y se varió la cantidad de unidades en la capa oculta entre 70, 80, y 90 unidades. Para el entrenamiento se generaron conjuntos de datos con 1000 instancias, de las cuales un 25 % fue reservado para validación. Los datos fueron generados al azar variando

la proporción de correlaciones entre variables como se especificó previamente. Dado el número reiterado de simulaciones bajo las mismas condiciones, el método de validación utilizado fue holdout, ya que ante la posibilidad de generar múltiples conjuntos de datos aleatorios no son necesarias más particiones para validación cruzada. Los parámetros utilizados con los algoritmos fueron para SGD $\eta = 0{,}001$, y para GDM, $\eta = 0{,}001$, $\beta = 0{,}5$.

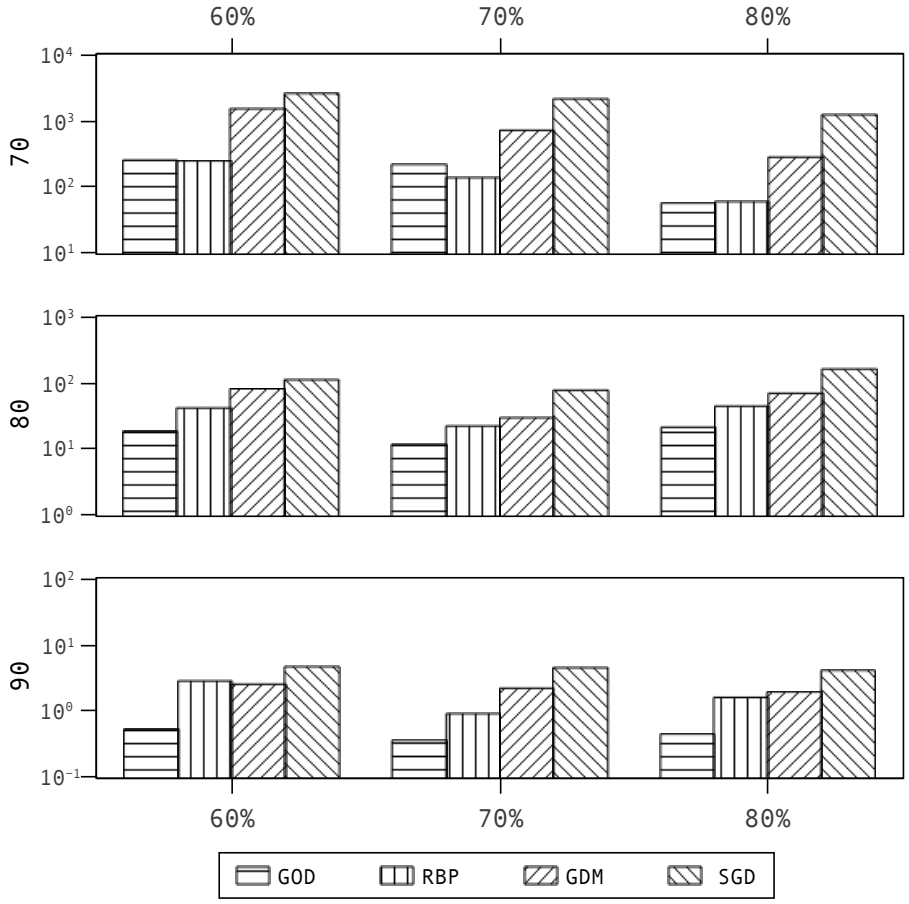

Figura 4.2.: Comparación de distintos algoritmos en problemas de auto-codificación con datos correlacionados.

En la figura Figura 4.2 se puede apreciar como varia la efectividad de los distintos algoritmos al variar la proporción de variables independientes y la cantidad de unidades en la capa oculta. Los tres gráficos corresponden a la variación de unidades en la capa oculta, indicadas por los valores a la izquierda (un valor menor corresponde a un problema más difícil), los porcentajes en el eje horizontal indican la proporción de variables dependientes utilizadas (un valor menor también corresponde a un problema más difícil), y los valores en los ejes verticales corresponden a la efectividad de los algoritmos calculada como ya fue indicado y promediada sobre varias corridas. En estas simulaciones se utilizaron 5 clases, los resultados para otra cantidad de clases no se muestran aquí ya

que solo presentan una variación no muy pronunciada pero acorde a la complejidad del problema.

Es de destacarse no solo cuanto varía la efectividad del algoritmo propuesto ante problemas de distinta complejidad, sino también como varia el comportamiento relativo de todos los métodos comparados. Dependiendo de la complejidad del problema la efectividad de esta solución varía relativamente con respecto a otros algoritmos, pero también en general se puede apreciar una clara tendencia a converger más rápidamente.

4.5. Conclusiones

El algoritmo propuesto funciona efectivamente para dos problemas típicos en la evaluación de esta clase de algoritmos, mostrando una tasa y velocidad de convergencia superior a otros algoritmos comúnmente usados. Se encontró también que en algunos casos, por ejemplo cuando el problema es muy sencillo, algoritmos más simples como SGD pueden ser más efectivos, pero a su vez tienen la desventaja de requerir una ajuste de parámetros para lograrlo. De la misma forma, con problemas que, por ejemplo por las limitaciones de la arquitectura, pueden ser imposibles, la efectividad de la solución propuesta puede ser comparable a la de RBP, pero a diferencia de este, el algoritmo presentado tiene la ventaja de poder funcionar en mini-lotes.

Parte de los contenidos y resultados presentados en este capítulo fueron previamente publicados en el artículo “Gradient Omissive Descent is a minimization algorithm” como parte de International Journal on Soft Computing, Artificial Intelligence and Applications 8, no. 1, 37–45. (2019)

5. Codificación de Lenguajes Libres de Contexto

5.1. Introducción

El modelo propuesto en este trabajo fue concebido para poder codificar secuencias de longitud variable en vectores de dimensión fija, de manera que esta codificación sea consistente con el contenido de la secuencia original. En particular, si el objetivo es trabajar en el procesamiento de lenguajes naturales es deseable saber en qué grado el modelo es capaz de codificar secuencias de acuerdo a su información semántica. Para esto debería contarse con una definición más o menos concreta de semántica, pero sería muy difícil poder dar una medida objetiva de ésta dada la naturaleza subjetiva del lenguaje.

En este capítulo se explora la relación entre algunos principios de la semántica cognitiva con la interpretación adoptada de semántica y los métodos utilizados por el modelo para producir codificaciones de secuencias. Además se propone el uso de un conjunto de sentencias generadas artificialmente, de modo que puedan ser fácilmente categorizadas de acuerdo a su significado a pesar de diferentes variaciones sintácticas, para análisis semánticos sobre palabras y sentencias.

5.2. Motivación

Dentro de este trabajo, el alcance de la interpretación de *semántica* estará limitado exclusivamente al *significado* del lenguaje escrito, en particular al *significado* de palabras, frases, o sentencias.

Así mismo, la interpretación de *significado* estará dada a partir de las hipótesis de trabajo provenientes de la semántica cognitiva. Esto es, la Hipótesis Distribucional (HD) dice que las palabras que tienden a aparecer en contextos similares tienden a tener *significado*s similares, y la Teoría de Prototipos (TP) dice que los distintos *significados* pertenecen de forma graduada a categorías dependiendo de su similitud con un prototipo [119, 35, 109].

En el caso de la HD, es fácil verificar su interpretación si se consideran ejemplos como los sinónimos, en donde distintas palabras pueden aparecer en el mismo contexto sin cambiar el significado general, o la polisemia, en donde una misma palabra puede variar su significado dependiendo del contexto en el que se encuentre. Para el caso de la TP,

en donde los significados pueden representarse como grados de semejanza a categorías, puede considerarse por ejemplo el caso del concepto "pingüino", en donde en parte la diferencia en un atributo como "alado" serviría para alejarlo de la idea de un pájaro prototípico [78].

Estas interpretaciones de *significado* pueden ser extendidas más allá de su aplicación a palabras y además ser asociadas con el funcionamiento del modelo propuesto.

Esto es, de la misma forma en que las palabras se combinan en frases para formar nuevos significados, es posible extender la idea de la HD y pensar que grupos de palabras que tienden a aparecer en los mismos contextos tienden a tener los mismos significados. A su vez es posible también pensar en los contextos como en combinaciones de palabras con un significado propio asociado.

La forma de trabajo del modelo propuesto está directamente relacionada con esta idea. En primer lugar porque cada codificación que produce, aún en las etapas intermedias, corresponde a una frase o subsecuencia de la entrada. Pero además porque estas codificaciones son producidas a partir de codificaciones correspondientes a dos subsecuencias superpuestas, de modo que el final de una subsecuencia coincide con el comienzo de la otra. El segmento común a las dos subsecuencias puede ser considerado como la frase cuyo significado se busca representar y los otros dos segmentos propios de cada subsecuencia, correspondientes a las porciones anteriores y posteriores, como el contexto en el que la frase ocurre.

Estas codificaciones son representadas por vectores cuyas dimensiones pueden ser interpretadas como distintos atributos, ya que son producidas por un tipo de activación dispersa. En este tipo de codificación se busca que solo una porción de las unidades neuronales estén activas simultáneamente, de modo que puedan representar las características más destacadas de los datos.

Esto coincide con la idea de la TP, ya que la tendencia de distintos grupos de neuronas a activarse para codificar distinta información hace que las representaciones tiendan a agruparse en cúmulos[1] análogos a prototipos de significado [69]. Además las correlaciones existentes gracias a la superposición de las subsecuencias facilitan esta codificación.

Como este mismo proceso se repite progresivamente en las sucesivas etapas, cada vez para subsecuencias mayores, en la última etapa se obtiene una codificación para la sentencia completa. Y como las codificaciones se obtienen aplicando siempre estos mismo principios, puede suponerse que las representaciones finales de las sentencias conformarán un espacio de representación semántica, en donde sentencias con significados similares obtendrán representaciones similares [79].

Sin embargo, para poder verificar la validez de estas codificaciones debería contarse además con algún tipo de representación más objetiva, de manera que se pueda estudiar la correlación entre ambas. Para esto se propone el uso de una Gramática Libre de Contexto (GLC) que pueda generar sentencias válidas en un lenguaje natural pero acotadas a un dominio específico. La GLC tiene la virtud de permitir, mediante la aplicación de distintas

[1] En inglés "clusters".

reglas gramaticales, la construcción de sentencias que se diferencien sintácticamente en distintos grados. A su vez, es posible también utilizar las reglas gramaticales elegidas para construir una representación semántica de las sentencias que servirá a modo de referencia o ground-truth.

5.3. Construcción de la Gramática

La gramática construida describe, en términos generales, situaciones comunes que podrían ocurrir en dibujos animados clásicos de las décadas del 40 y 50. Esto permite entender fácilmente las situaciones descriptas por las sentencias dado que es un material con el que casi todos estamos bastante familiarizados. En la misma se incluyeron reglas que contuvieran sinónimos y construcciones sintácticamente distintas pero semánticamente equivalentes. La especificación completa de las reglas de la gramática se incluye en el *Apéndice A*; a continuación se describen la motivación y los detalles de su construcción.

En principio es capaz de generar tres tipos de sentencias: unas para indicar frases con una acción (Faccion), otras para frases en las que un personaje vigila una acción (Fvigilia), y finalmente algunas en la que dos frases de acción ocurren simultáneamente (Fsimul). Como convención se adopta que los símbolos no terminales comienzan con mayúsculas.

S $\rightarrow$ Faccion | Fvigilia | Fsimul

Faccion $\rightarrow$ FVa | FVa IL | IL FVa

Fvigilia $\rightarrow$ Personaje Vvigilia FVa

Fsimul $\rightarrow$ FVa Vsimul FVa

Las frases de acción incluyen una frase con verbo de acción (FVa) y dos variantes que pueden incluir un Instrumento y Lugar (IL) relacionados con la acción. Las frases de verbo con acción a su vez pueden ser de dos tipos, de persecución (FVpersigue) o de agresión (FVgolpea).

FVa $\rightarrow$ FVpersigue | FVgolpea

FVpersigue $\rightarrow$ Perro Persigue Gato | Gato Persigue Raton | Gato Persigue Canario | Perro Persigue Cartero | Raton Persigue Señora | Gato EsPerseg Perro | Raton EsPerseg Gato | Canario EsPerseg Gato | Cartero EsPerseg Perro | Señora EsPerseg Raton

Persigue $\rightarrow$ *persigue a* | *corre detras de*

EsPerseg $\rightarrow$ *es perseguido por* | *corre delante de*

Las frases de verbo de persecución introducen ciertas restricciones en la forma en la que esta acción puede aparecer. Por ejemplo, el perro solo puede perseguir al gato o al cartero, el gato solo puede perseguir al ratón o al canario, y el ratón solo puede perseguir

a la señora. Al no ser válidas todas las combinaciones posibles estamos definiendo ciertas características sobre los personajes.

Además para describir la acción se utilizan tanto la voz activa y pasiva, como también algunos sinónimos. Por ejemplo las sentencias "el gato persigue a un ratón", "el gato corre detrás de un ratón", "un ratón es perseguido por el gato", y "un ratón corre delante de el gato" están todas describiendo la misma situación de distintas formas.

FVgolpea → Agresor Golpea Gato | Gato Golpea Victima | Gato EsGolp Agresor | Victima EsGolp Gato

Golpea → *golpea a* | *le pega a*

EsGolp → *es golpeado por* | *recibe un golpe de*

Agresor → Perro | Raton | Señora

Victima → Raton | Canario | Humano

Humano → Señora | Cartero

Animal → Perro | Gato | Raton | Canario

Para las frases de acción de agresión se introducen también restricciones, pero distintas de las de persecución. En este caso se toma al Gato en cierta forma como un pivote y se definen Agresores y Víctimas del mismo. Además se introduce un nuevo nivel de complejidad al incorporar al Ratón y a la Señora tanto como víctimas y agresores del Gato. En este caso también se hace uso de la voz pasiva, activa y sinónimos para describir variantes de la mismas acción.

Perro → ArtM *perro* | *butch* | *spike*

Gato → ArtM *gato* | *tom* | *silvestre*

Raton → ArtM *raton* | *jerry*

Canario → ArtM *canario* | *tweety*

Señora → *la señora*

Cartero → *el cartero*

ArtM → *el* | *un*

ArtF → *la* | *una*

Para los personajes se agregan nuevas equivalencias en donde varios sujetos pueden ejecutar los mismos roles. Por ejemplo "el gato", "un gato", "tom", y "silvestre" son completamente equivalentes en relación a las situaciones en las que pueden aparecer. Esto significa que desde el punto de vista del lenguaje generado por la gramática son sinónimos.

Además, las frases con verbos de acción tienen dos formas posibles en las que se puede indicar en que lugar ocurre la acción, con que instrumento, o ambos. El complemento

de Lugar puede tomar dos formas, en la casa (Lcasa) y en la calle (Lcalle). A su vez dentro de la casa puede hacer referencia a la cocina (Lcocina), la sala (Lsala), o el patio (Lpatio). En los casos en los que se hace referencia a ambos complementos, de lugar e instrumento, para cada parte se puede formar la frase de dos formas equivalentes, como "con Instrumento en Lugar" o como "en Lugar con Instrumento".

IL → Lugar | *con* Instrumento | *con* Icocina Lcocina | Lcocina *con* Icocina | *con* Isala Lsala | Lsala *con* Isala | *con* Ipatio Lpatio | Lpatio *con* Ipatio | *con* Icalle Lcalle | Lcalle *con* Icalle

Lugar → Lcasa | Lcalle

Lcasa → PLE *la casa* | Lcocina | Lsala | Lpatio

Lcalle → PLE *la calle* | Pcalle

Lcocina → PLE *la cocina* | Pcocina

Lsala → PLE *la sala* | Psala

Lpatio → PLE *el patio* | Ppatio

Adicionalmente se pueden especificar Partes de Lugares Generales (PLG) o Específicos (PLE) de cada lugar. Por ejemplo para la cocina se puede hacer referencia a la heladera, la mesa, o la ventana, en la calle se pueden encontrar el tacho y el poste, etc. Todas estas reglas permiten formar una geografía del lugar que queda implícitamente definida por las sentencias.

Pcocina → PLG *la heladera* | PLG *la mesa* | PLG *la ventana*

Psala → PLG *el sillon* | PLG *la chimenea* | PLG *la puerta*

Ppatio → PLG *el jardin* | PLG *el arbol* | PLG *la cucha*

Pcalle → PLG *el tacho* | PLG *el poste*

PLE → *en* | *dentro de* | PLG

PLG → *por* | *junto a* | *delante de*

Finalmente, cuando se hace referencia a un instrumento y un lugar en la misma sentencia se introduce la restricción sobre qué instrumentos pueden ser usados en qué lugares. Esta restricción no es única para cada instrumento, ya que algunos pueden ser utilizados en varios lugares. Esto contribuye a agregar algunas características adicionales sobre los elementos que aparecen en las sentencias y que van más allá de su mero carácter sintáctico.

Instrumento → Icasa | Icalle

Icasa → Icocina | Isala | Ipatio

Icocina → *la escoba* | *la botella* | *la sarten* | ArtM *jarron*

Isala → *la escoba* | *el florero* | *el atizador*

Ipatio → *la escoba* | *el ladrillo* | *el martillo* | ArtF *pala*

Icalle → *la botella* | *el ladrillo* | *el fierro* | ArtM *palo*

Los otros dos tipos de sentencias que pueden ser generados, de acciones vigiladas y simultáneas, simplemente agregan unas reglas muy sencillas que permiten extender el tipo de situaciones que describen las frases de acción. Por ejemplo la frase de acción "spike le pega a tom" puede ser extendida como "el cartero observa como spike le pega a tom".

Fvigilia → Personaje Vvigilia FVa

Vvigilia → *observa como* | *mira como* | *sabe que* | *sospecha que*

Personaje → Humano | Animal

E inclusive se pueden llegar a describir dos acciones en una misma sentencia como "tweety es perseguido por un gato mientras que silvestre es golpeado por la señora".

Fsimul → FVa Vsimul FVa

Vsimul → *mientras que* | *cuando*

Con esto queda definida una gramática de 129 reglas, que es capaz de generar más de 900 mil sentencias distintas utilizando solo 70 palabras. Si bien la gramática no contiene reglas recursivas igualmente puede formar sentencias complejas que describen un amplio rango de situaciones.

5.4. Arquitectura y Entrenamiento del Modelo

En primer lugar, para poder utilizar las sentencias generadas por la gramática, se debe elegir algún tipo de codificación inicial para que pueda ser alimentada a la entrada del modelo. En este caso las palabras fueron codificadas en vectores con 8 valores bipolares[2] aleatorios, de modo que su codificación no reflejara ni características sintácticas ni semánticas y fueran suficientemente diferentes entre sí.

Para el entrenamiento se utilizó un conjunto de 10000 sentencias generadas aleatoriamente a partir de la gramática. Para la validación y las pruebas experimentales se generaron otros tres conjuntos de 1000 sentencias adicionales. Estos conjuntos fueron convertidos a secuencias de vectores utilizando las codificaciones de palabras previamente establecidas.

Para poder determinar experimentalmente la arquitectura del modelo, en particular la cantidad de unidades por etapa, se adoptó como criterio que las dimensiones elegidas

[2]Compuestos por -1 y +1. Similar a referirse a vectores compuestos por 0 y 1 como binarios.

fueran las mínimas necesarias para preservar suficiente información característica de las secuencias, pero no necesariamente tanta como para poder hacer una reconstrucción sin error. Esto tiene la ventaja de evitar un incremento en la cantidad de unidades requeridas en las últimas etapas, necesario para hacer una reducción efectiva en la dimensión dada la naturaleza aleatoria en la codificación inicial de las palabras.

La estrategia utilizada se basó en un tipo de búsqueda del intervalo medio, similar a un método de búsqueda binaria o por bisección, entre la misma cantidad de unidades de la etapa anterior y el doble. Adoptar el mismo tamaño que la etapa anterior implicaría que los pares de patrones a codificar están completamente correlacionados y pueden ser representados en el espacio dimensional de uno, adoptar el doble del tamaño implicaría que no existe ninguna correlación y no es posible reducir la dimensión de estos pares.

Para verificar que la cantidad de unidades elegidas en cada etapa era la adecuada se entrenaron sobre el mismo conjunto de datos tres redes distintas pero de idéntica arquitectura y se testeó cada red con cada uno de los tres conjunto de datos de validación. Si una cantidad mayoritaria de combinaciones cruzadas presentaba un error promedio menor a 0.1 se consideraba que una etapa con esta cantidad de unidades podía aprender y generalizar aceptablemente. Finalmente se creaba una red con la cantidad de unidades elegidas y se la entrenaba con los conjuntos de datos combinados para generar las secuencias de la siguiente etapa. El valor límite de 0.1 puede parecer poco estricto, pero en este caso funciona en forma similar a la parada-temprana (early-stopping), previniendo los sobre-ajustes.

Finalmente la arquitectura determinada por este método produjo un modelo con 7 etapas dimensionadas en 12, 16, 21, 25, 37, 57, y 86 unidades. Las primeras 3 etapas utilizan un paso de 1, incorporando suficiente redundancia en la codificación, mientras que el resto de las etapas posteriores utiliza un paso de 2. Esto permite codificar sentencias de longitud máxima de 19 palabras. Cada una de estas etapas consiste en un auto-codificador en la forma de un perceptrón multicapa con una sola capa oculta cuya activación depende del método de codificación dispersa propuesto (Ecuación 3.8) [62] y, entrenado utilizando el algoritmo de gradiente omisivo descripto anteriormente (Algoritmo 4.1) [40].

5.5. Agrupaciones Jerárquicas de Palabras por Semántica

Uno de los aspectos que se desea estudiar es la capacidad del modelo para representar el significado de las palabras a partir de como éstas son usadas en las sentencias. Esto implica que palabras con significados similares deberían obtener representaciones similares. Sin embargo, rescatar la representación de una palabra individual entre las codificaciones hechas por las progresivas etapas intermedias del modelo no es sencillo.

Una posible estrategia para tratar de capturar esta información podría consistir en generar una secuencia compuesta por un vector que represente a una palabra, precedido y

sucedido por varios patrones en blanco, y estudiar la codificación producida por la etapa en donde esta secuencia sea reducida a un solo patrón. Por ejemplo, considerando una secuencia de la forma [_ , _ , w , _ , _], en donde w es el vector que representa a la palabra y los guiones bajos representan a los patrones en blanco, se podría obtener una representación de un solo patrón en la tercera etapa. No obstante, a pesar de que los patrones en blanco representan una suerte de contexto genérico en el que la palabra puede aparecer, esta estrategia no parece capturar suficiente información como para que los resultados sean apreciables.

Otra alternativa podría ser codificar directamente a la palabra como una secuencia de un solo elemento utilizando todo el modelo. Sin embargo, con esta técnica los resultados obtenidos son todos muy similares, independientemente de la palabra utilizada. El problema se debe a que la información de la palabra se va perdiendo en la propagación por las etapas en una situación similar a la de relación señal-ruido. Es de esperarse que ocurra algo así dado que el modelo fue entrenado con sentencias de unas diez palabras en promedio aproximadamente.

Finalmente, la codificación se realizó tomando la salida final del modelo a partir de una secuencia generada con la repetición de la misma palabra, de forma similar a un *mantra*. Esto no solo hace que la información de la palabra se mantenga a través de las etapas, sino que además se refuerce progresivamente. Las sentencias generadas de esta manera estuvieron compuestas por una misma palabra repetida 19 veces, que es la longitud máxima de secuencia que este modelo puede codificar.

Para la prueba se eligió un grupo de sustantivos de los más utilizados en los conjuntos de datos y que pertenecieran a los grupos de personajes, lugares e instrumentos, tratando de tomar una cantidad representativa de cada grupo. Sintácticamente todas estas palabras pertenecen a la misma categoría, son sustantivos. Pero si el modelo tiene capacidad para representar información semántica los proyectará a regiones distintas del espacio de salida. Sobre todo teniendo en cuenta que, por la forma en la que fue construida la gramática, estas palabras suelen aparecer en roles muy diferentes y específicos.

Para analizar los resultados se recurrió a un dendrograma con las codificaciones resultantes de las palabras. Un dendrograma es un tipo de diagrama en forma de árbol formado por agrupaciones jerárquicas, en este caso, de vectores [125]. Una forma posible de construirlo es tomando la distancia euclidiana entre todos los puntos y eligiendo los dos más próximos para formar un grupo. Este grupo se agrega como un nuevo nodo en el árbol que será representado por un nuevo punto, por ejemplo tomando el promedio de los puntos agrupados. Este proceso se vuelve a repetir hasta que quede un solo nodo.

En la figura Figura 5.1 se puede apreciar que la codificación distingue entre los personajes, lugares e instrumentos. Sin embargo se debe notar también que las distinciones entre lugares e instrumentos no son tan claras. Esto probablemente sea debido a que se diversificó demasiado sobre estos complementos en la gramática, teniendo en cuenta el número de ejemplos utilizados como datos.

Otro problema es que en los subgrupos, dentro de las tres categorías principales, no llega a producirse el nivel de auto-ordenamiento esperado. Por ejemplo uno esperaría

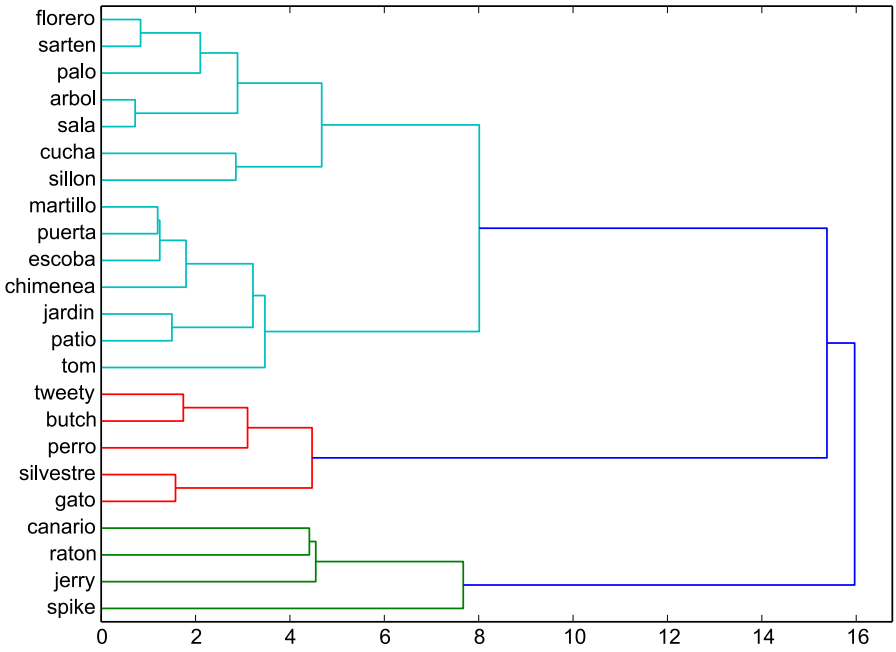

Figura 5.1.: Agrupación jerárquica en forma de dendrograma entre las distancias de codificación producidas para secuencias consistentes en la repetición de sustantivos.

que *gato*, *tom* y *silvestre* fueran más cercanos, o al menos, con un agrupamiento más marcado entre perseguidores o agresores.

No obstante, debe tenerse en cuenta que es difícil interpretar estos resultados directamente ya que el modelo no fue concebido para codificar palabras individuales o secuencias compuestas por la misma palabra repetida, y el objetivo principal de esta prueba es solamente poder mostrar una aplicación similar a los word-embeddings. Teniendo en cuenta esto, se puede considerar que los resultados son aceptables ya que, a pesar de los inconvenientes señalados, la codificación de las palabras tiende a un agrupamiento semántico [121, 77, 49].

5.6. Mapeo de Sentencias a un Espacio Semántico

La capacidad del modelo para codificar palabras individuales de acuerdo a su significado es una característica importante, pero el fin último que se está persiguiendo es la codificación de nuevos significados cuando las palabras se combinan para formar sentencias. Esto es, si las sentencias son codificadas por el modelo a una representación dentro de un espacio semántico, entonces se debería cumplir que sentencias con significados similares tiendan a ser mapeadas o proyectadas a puntos cercanos dentro de este espacio de

representación [110, 36].

Para los lenguajes naturales en general es muy difícil poder establecer objetivamente el grado de similitud semántica entre sentencias, pero en este caso la gramática fue elegida de modo tal que facilitara la creación de una categorización semántica de acuerdo a las reglas aplicadas al construir las sentencias. De esta forma se asegura que sentencias con significados similares obtengan una categorización semántica similar o idéntica.

La representación de la categorización semántica para las sentencias consiste en un vector binario de dimensión 26, dividido en 5 bits para cada uno de los dos personajes intervinientes, 2 bits para el tipo de acción involucrada, 8 bits para el complemento de lugar y 6 bits para el complemento de instrumento.

Las representaciones fueron elegidas tratando de producir ortogonalidad entre clases pero adoptando un valor igual para todos los miembros pertenecientes a la misma categoría. Por ejemplo las codificaciones para perros y gatos son 01000 y 00100 respectivamente, pero las representaciones para todas las variantes de "perro" (un/el perro, butch, spike) son idénticas (01000).

Así mismo las frases en donde existe algún tipo de variante que no altere el significado, como el uso de sinónimos, voz pasiva o cambios en el orden del complemento circunstancial, también reciben idéntica representación.

Para las sentencias en la que un tercer personaje vigila la acción, la representación elegida para la sentencia consiste en promediar la representación de éste con la de los otros dos personajes involucrados. En el caso de las sentencias que describen dos acciones simultáneas, la representación elegida es la correspondiente al promedio de las representaciones de las dos frases de acción involucradas.

El criterio detrás de todas estas formas de representación no es solamente ocultar las diferencias sintácticas entre sentencias con significados similares sino que también, sentencias que sean semánticamente muy similares, en lugar de recibir representaciones cercas, recibirán una representación con idéntica categorización.

Por ejemplo las sentencias "`spike persigue a tom`" y "`silvestre corre delante de butch`" son sintácticamente muy distintas pero están describiendo situaciones muy similares por lo que a ambas les corresponde el mismo vector de categoría semántica.

El fin de estas representaciones es que puedan ser consideradas como ground-truth y esto permita estudiar el grado de correlación entre ellas y las codificaciones de sentencias producidas por el modelo. Al considerar sentencias sintácticamente distintas pero con significados similares dentro de una misma categorización semántica se está imponiendo un grado más alto de dificultad al problema para así tratar de asegurar una mayor robustez del modelo [42, 113]. Es importante también notar que estas representaciones elegidas para las sentencias no guardan ninguna relación con las codificaciones aleatorias de las palabras elegidas inicialmente para el entrenamiento del modelo.

Para la experimentación se construyeron dos conjuntos, cada uno de 1000 sentencias elegidas al azar sin repetición. Para cada una de estas sentencias se obtuvieron las

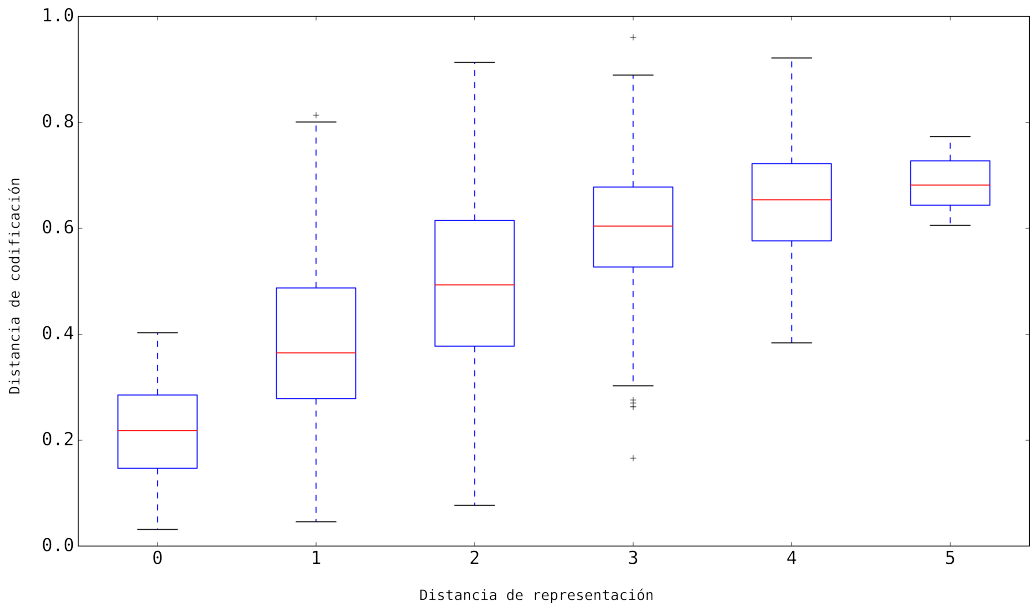

Figura 5.2.: Para pares de sentencias se muestra la norma L_2 de las diferencias entre las codificaciones producidas por el modelo en función de las de representación de categorización semántica.

codificaciones producidas por el modelo, conjuntos C_1 y C_2, y las representaciones de categorización semántica correspondientes, conjuntos R_1 y R_2.

Para cada codificación del primer conjunto (C_1) se buscó la codificación dentro del segundo conjunto (C_2) que haya recibido la codificación más próxima, es decir, cuya diferencia tenga la menor norma L_2. Para los mismos pares de sentencias se calculó de forma análoga la distancia entre sus representaciones de categorización semántica. Este par de valores fue finalmente utilizado para definir el conjunto de coordenadas que relacionan la distancia de codificación con la de representación.

$$D = \{(\|r_i - r_j\|, \|c_i - c_j\|)/\forall c_k \in C_2(\|c_i - c_j\| \leq \|c_i - c_k\|)\}$$

En donde $r_i \in R_1, r_j \in R_2, c_i \in C_1, c_j \in C_2$ y los subíndices como i en r_i y c_i indican la representación y codificación correspondientes a la misma sentencia.

En la Figura 5.2 se pueden apreciar las distancias de codificación normalizadas en función de las distancias de representación para los mismos pares de sentencias. El gráfico muestra la mediana, el primer y tercer cuartil, variabilidad en el rango intercuartil 1.5x y eventuales valores atípicos. Esta representación fue elegida porque da un resumen visual de la tendencia, simetría y dispersión de los datos, como una demostración más descriptiva que utilizando puntos de valores individuales.

Sobre el conjunto de las coordenadas originales se observa un coeficiente de Pearson de 0.585 indicando una clara correlación entre la codificación de sentencias producida

por el modelo y la representación de categorización semántica. Esto significa que cuanto más semánticamente similares son dos sentencias, de acuerdo a la representación de significado elegida, tenderán a ser codificadas por el modelo a vectores más cercanos. Si bien no fue posible obtener un mayor grado de correlación, debido principalmente a una más amplia dispersión en los valores medios, estos resultados pueden ser considerados aceptables teniendo en cuenta que la representación inicial de las palabras fue elegida de manera completamente aleatoria, es decir que no contienen ninguna información semántica, sintáctica, o morfológica, y que la representación de la categorización semántica de las sentencias fue elegida para imponer condiciones más estrictas para el problema.

0.0	0.063	silvestre observa como butch persigue a silvestre.
		tom observa como butch persigue a tom.
1.4	0.221	silvestre corre detras de un raton con la botella por el tacho.
		silvestre corre detras de un canario con la botella por el tacho.
2.2	0.406	delante de la cocina con el jarron jerry corre delante de un gato.
		dentro de la cocina con el jarron un gato le pega a la senora.
3.0	0.601	tom golpea a el perro junto a el jardin con la escoba.
		tom persigue a el raton junto a la ventana con el jarron.
4.5	0.821	un gato le pega a el raton cuando la senora le pega a un gato.
		el cartero es perseguido por el perro con la escoba por el patio.

Cuadro 5.1.: Ejemplos de pares de sentencias con las correspondientes distancias entre representación (primera columna) y codificación (segunda columna).

En la Tabla 5.1 también se pueden apreciar algunos ejemplos concretos de pares de sentencias con sus correspondiente distancias entre codificaciones producidas por el modelo y representaciones de categorización.

La interpretación de estos resultados indica que el modelo es capaz de codificar la información semántica de las sentencias a partir de la estrategia propuesta basada en la Hipótesis Distribucional y a la Teoría de Prototipos, y que sentencias con significados similares serán proyectadas a puntos cercanos dentro del espacio de representación semántico [74, 50]. Esto es, que la estrategia de codificación progresiva de pares de patrones puede capturar *significado* de acuerdo a lo propuesto por la Hipótesis Distribucional, no solo para palabras sino también para frases y sentencias, y que este *significado* puede ser representando según lo propuesto por la Teoría de Prototipos por una codificación dispersa obtenida mediante auto-codificadores que pueden ser entrenados eficazmente.

5.7. Conclusiones

Partiendo de la noción de significado de palabras propuesto por la Hipótesis Distribucional y la Teoría de Prototipos se pudo ofrecer una forma de extender el uso de estos

mismos principios para la interpretación del significado de frases y sentencias. Esto a su vez fue relacionando con las estrategias que utiliza el modelo propuesto para obtener codificaciones vectoriales de sentencias.

Mediante el uso de un conjunto de sentencias generadas por una Gramática Libre de Contexto se pudo verificar además que el modelo es capaz de producir codificaciones que conformen un espacio de representación semántica. Esto fue posible gracias a una categorización semántica de las sentencias utilizada como referencia para determinar el grado de correlación entre ambas. Esta categorización puede ser obtenida fácilmente a partir de las reglas gramaticales involucradas en la generación de cada sentencia.

A pesar de que el conjunto de datos artificiales no posee el mismo tipo de variedad y complejidad encontrado en sentencias pertenecientes a datos reales, pero dadas las condiciones experimentales más estrictas, como la representación de variaciones sintácticas en igual categoría semántica o el uso de codificaciones aleatorias para las palabras en los datos, es posible estimar que las estrategias utilizadas pueden ser también válidas con otros datos provenientes de lenguajes naturales.

6. Reconstrucción de Secuencias y Morfología

6.1. Introducción

El modelo presentado en este trabajo es capaz de codificar secuencias de longitud variable en vectores de dimensión fija aprendiendo sobre las regularidades que existen en los patrones dentro de las secuencias [41]. Estas regularidades pueden ser explotadas porque, en principio, no todas las combinaciones posibles son válidas, y porque diferencias en la frecuencia de aparición hace que las probabilidades entre todas la combinaciones válidas no sean iguales. Es decir, hay secuencias o subsecuencias que son más probables que otras.

Si éstas regularidades están gobernadas por estructuras semánticas entonces el modelo aprenderá sobre semántica, si están gobernadas por estructuras morfológicas entonces aprenderá sobre morfología. En este caso, el problema de codificar representaciones morfológicas, considerando las secuencias de símbolos como palabras formadas por secuencias de letras, es considerablemente más sencillo que su equivalente semántico. No sólo porque la combinación de morfemas en los lenguajes occidentales es notablemente regular [129], sino porque es relativamente razonable considerar la posibilidad de tratar con un conjunto de datos compuesto por un diccionario que cubra una proporción casi completa de las palabras posibles, al menos limitadas al lenguaje de uso cotidiano.

Esto hace que el problema sea especialmente adecuado para estudiar el tipo de representación generada por el modelo, no solo teniendo en cuenta su capacidad de codificar secuencias a vectores, sino también la de producir secuencias a partir de la decodificación de vectores.

6.2. Propiedades de la Representación

Teniendo en cuenta que la información morfológica es suficientemente regular y que es posible trabajar con lo que podría considerarse un conjunto de datos relativamente completo, esto presenta la oportunidad de estudiar el comportamiento del modelo y las propiedades de las representaciones generadas cuando se busca que la codificación producida conserve suficiente información como para que sea posible reconstruir la secuencia original.

Esto depende en gran parte del tipo de representación que se obtenga al codificar la secuencia [117, 45], y en particular podría pensarse en dos propiedades especialmente deseables.

La primera, a la que llamaremos *unicidad*, en relación a la correspondencia entre cada elemento de un conjunto con un único elemento de otro, servirá para indicar que secuencias distintas deben obtener representaciones distintas, o en un sentido más específico, cada secuencia válida debe obtener una representación suficientemente distinta como para que sea posible diferenciarla. Esto es necesario ya que, si se desea poder decodificar la secuencia original a partir de la representación obtenida en la codificación, esta representación no puede ser ambigua. Pero esto no es suficiente, de lo contrario un simple índice sobre las secuencias válidas podría ser aceptable, es además necesario que esta representación contenga información sobre la composición de la secuencia [16].

Esto lleva a la segunda propiedad, a la que llamaremos *consistencia*, en relación a la cercanía entre codificaciones para secuencias semejantes, y servirá para indicar que secuencias que compartan alguna subsecuencia en común deberían obtener al menos codificaciones parcialmente similares. Esto sirve para que las representaciones obtenidas sean regulares, es decir, que secuencias similares sean codificadas a representaciones similares, y la hace además necesaria para poder generalizar al resto del lenguaje [78].

En definitiva entonces, lo que se busca es un tipo de codificación que produzca representaciones similares para secuencias similares, pero suficientemente distintas para secuencias válidas diferentes.

Para estudiar estas propiedades se seleccionaron dos tipos de conjuntos de datos, de tamaño menor (1000 palabras), y mayor (20 mil palabras), y se determinaron experimentalmente arquitecturas de modelos para ser entrenados con estos conjuntos distintos. Los datos estuvieron compuestos por diccionarios de palabras pertenecientes al idioma inglés, cada palabra fue considerada como una secuencia de símbolos consistentes en letras mayúsculas y minúsculas. A su vez, las letras fueron convertidas a patrones utilizando la representación binaria del correspondiente código ASCII, de modo que cada símbolo de la secuencia quedó representado por un vector de dimensión 8 compuesto únicamente por valores -1, o $+1$. Los tamaños de las etapas posteriores fueron determinados utilizando un algoritmo de búsqueda del intervalo medio entre un tamaño igual y del doble del de la etapa anterior, ya descripto anteriormente (Sección 5.4). Para asegurar la posibilidad de reconstrucción de secuencias la minimización del error se evaluó tanto para los datos de entrenamiento como los de validación durante la búsqueda del tamaño de dimensión óptimo. En todos los casos la arquitectura elegida estuvo compuesta por 7 etapas capaces de codificar secuencias de hasta 19 símbolos. Las dimensiones en cada etapa variaron de acuerdo al tamaño del conjunto de datos utilizado.

Consideremos el caso de un modelo entrenado con un conjunto de 1000 palabras cuya codificación final consistió en un vector de dimensión 128 con valores acotados entre -1 y $+1$. Si tomamos las representaciones obtenidas para este conjunto de datos es posible ver a que palabras corresponden las codificaciones que presentaron las menores distancias vectoriales entre ellas. En la figura Figura 6.1 se puede apreciar un esquema de las

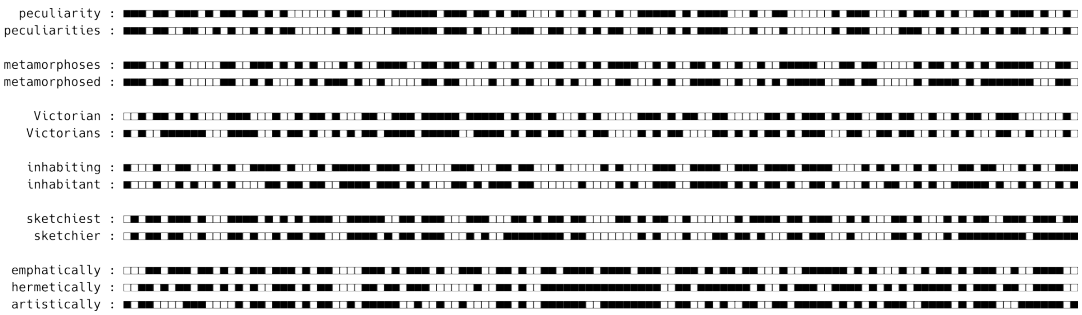

Figura 6.1.: Palabras del conjunto de datos con las representaciones de vectores similares.

representaciones obtenidas en donde los valores positivos son representados con cuadros negros y los negativos con cuadros blancos. Aquí se puede ver como palabras con la misma raíz pero pertenecientes a distintas categorías gramaticales (como variaciones en número, tiempo, etc), como también palabras en la misma categoría pero de distinta raíz, obtienen representaciones notablemente similares pero suficientemente distintas como para poder diferenciarlas [79, 47]. Es importante notar que estas representaciones no tienen, ni intentan tener, ninguna información sobre el significado de las palabras. Es una representación puramente morfológica, no semántica. Sin embargo, aún así, esto no es suficiente para afirmar que la codificación cumple con las propiedades deseadas. Para poder analizar hasta que grado se verifican estas propiedades estudiaremos el comportamiento del modelo y de las representaciones que produce en relación a dos tipos de problemas específicos relacionados con la representación del conocimiento y la morfología en lenguajes naturales.

6.3. Unicidad y Memoria Auto-Asociativa

Existe un tipo de modelo conocido como Memoria Auto-Asociativa (Auto-Asociative Memory, AAM) que es capaz de memorizar un conjunto de patrones y que, cuando se le presenta una versión incompleta o distorsionada de uno de ellos, responde evocando el patrón más similar al estímulo presentado de entre los memorizados [51, 1]. Un problema directamente relacionado con las palabras en lenguajes naturales y que ilustra claramente la potencial aplicación de este tipo de modelos sería la identificación de errores tipográficos. En este caso la AAM memorizaría un diccionario de palabras correctamente escritas, y al ser presentado con alguna palabra que incluyera errores tipográficos debería ser capaz de evocar la palabra original. Podemos utilizar este problema para estudiar el alcance y eficacia de la *unicidad* en nuestro modelo.

Si consideramos al conjunto de entrenamiento como los patrones memoria, podemos pensar que para el modelo estas serán palabras a las que llamaremos *conocidas*. Otras palabras válidas del lenguaje pero que no fueron parte del conjunto de entrenamiento serán consideradas como palabras *nuevas*. Y también vamos a considerar un potencial

conjunto de palabras *erróneas* en la forma de variaciones de las palabras *conocidas* con al menos un error tipográfico. Los errores tipográficos a su vez pueden ser de cuatro tipos:

Agregar: Se agrega un símbolo al azar en una posición al azar de la secuencia.

Omitir: Se elimina un símbolo de una posición al azar de la secuencia.

Reemplazar: Se reemplaza el símbolo de una posición al azar por un símbolo al azar.

Permutar: Un símbolo es permutado por su símbolo consecutivo para una posición al azar.

Si bien los dos últimos tipos de errores podrían considerarse como casos particulares de la aplicación sucesiva de los dos primeros, por ser errores encontrados muy frecuentemente en datos reales, se los incluyó también como casos específicos.

Figura 6.2.: Representaciones vectoriales para palabras similares a "every".

Comencemos por analizar un caso particular. Las palabras "every", "ever", y "fever" fueron incluidas en el conjunto de entrenamiento y son consideradas como *conocidas*. En la figura Figura 6.2 se puede observar como se comparan sus codificaciones con algunas variantes *erróneas* de "every". Si comparamos las distancias entre estos vectores podemos ver que la distancia de las palabras erróneas tiende a ser menor que la que existe entre las palabras conocidas. Aquí utilizamos la notación $\vec{word}$ para indicar el vector correspondiente a la codificación de la palabra "word", y $\|\cdot\|$ para indicar la norma L2, o distancia euclidiana del vector resultante de la diferencia entre las dos representaciones.

$$\begin{aligned} \|e\vec{very} - e\vec{ver}\| &= 13.227 \\ \left\|e\vec{ver} - f\vec{ever}\right\| &= 13.239 \\ \left\|e\vec{very} - f\vec{ever}\right\| &= 13.631 \end{aligned}$$

$$\begin{aligned} \left\|e\vec{very} - e\vec{bery}\right\| &= 12.867 \\ \|e\vec{very} - e\vec{vry}\| &= 12.982 \\ \|e\vec{very} - e\vec{veryy}\| &= 13.695 \end{aligned}$$

Sin embargo esto no garantiza un comportamiento similar al de un AAM, y por lo general el resultado de reconstruir la codificación de una palabra errónea es la misma palabra errónea. El problema radica en la dualidad entre la capacidad para memorizar y la capacidad para generalizar. La arquitectura de este modelo fue concebida priorizando la capacidad para generalizar. Aún en un caso como el que estamos considerando, en donde un conjunto de entrenamiento de solo mil palabras es relativamente chico, en donde se

está permitiendo hasta cierto grado el sobreajuste de los parámetros, y limitándonos únicamente a errores simples, aún así, el modelo no tiene información suficiente para diferencia entre una palabra *errónea* y una *nueva*. Es decir, cuando se le pide que reconstruya una palabra con un error tipográfico, en lugar de corregir el error y reconstruir la palabra *conocida* original, en la mayoría de los casos reconstruye la misma palabra *errónea* porque la considera como una palabra *nueva*. Es posible mejorar los resultados implementando una estrategia de activación que involucre codificaciones y decodificaciones sucesivas hasta llegar a lo que podríamos llamar un atractor estable [51], pero las dificultades para elegir la arquitectura y los parámetros hacen que de todas formas esta siga siendo una solución impráctica.

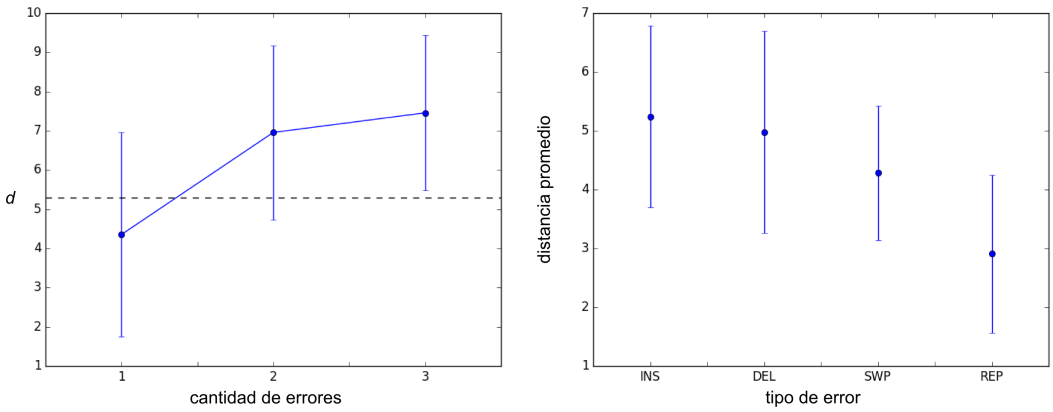

Figura 6.3.: Distancia promedio entre palabras conocidas (d) y entre la codificación de estas palabras con hasta tres errores tipográficos (izq). Distancia promedio para distintos tipos de errores simples (der).

Sin embargo este problema es ideal para verificar la *unicidad* comparando las distancias entre palabras *conocidas* y palabras *erróneas*. Para esto se consideró un modelo entrenado con un conjunto de 20 mil palabras además de dos conjuntos de prueba de mil palabras distintas elegidas al azar. Para cada palabra del primer conjunto de prueba se tomó la menor distancia entre las codificaciones de las palabras del segundo conjunto. El promedio de estas mínimas distancias es considerado como el límite para diferenciar palabras conocidas entre sí. A cada una de las palabras del primer conjunto se le aplicaron 3 errores tipográficos sucesivos al azar, calculando la distancia entre la palabra original y la codificación con errores. En la figura Figura 6.3 se puede apreciar que las palabras *erróneas* tienden a tener una codificación más cercana que otras palabras *conocidas* cuando se trata de errores simples. Además se puede ver que los tipos de error influyen de forma diferente, y que los errores que modifican la longitud de la palabra producen en promedio codificaciones más distintas.

Pero para contar con algún tipo de evidencia que sea más estadísticamente significativa fue necesario además buscar un tipo de medida numérica que permita comparar la semejanza entre palabras para poder aplicarlo fácilmente a cientos de pares.

La medida elegida para la comparación fue la Relación de Pareo de Secuencias, o Sequence Matching Ratio (SMR) que, dadas dos palabras, dice en qué proporción estas coinciden basándose en las subsecuencias comunes continuas más largas [97].

Estos valores fueron comparados con la Distancia Vectorial Normalizada (DVN) entre las codificaciones producidas por el modelo para los mismos pares de palabras. Cabe señalar que las dos medidas tienen una relación inversamente proporcional, dado que para palabras iguales el SMR es 1 y la DVN es 0.

Los datos utilizados en la experimentación fueron 500 pares de palabras en inglés seleccionados al azar de un diccionario de unas 70 mil palabras. La selección de los pares de palabras se realizó tratando de formar un conjunto balanceado de acuerdo al SMR. Es decir, la probabilidad de elegir un par de palabras disminuía de acuerdo a la cantidad de pares elegidos cercanos a su SMR . Esto fue necesario debido a que eligiendo de forma aleatoria aparecen muchos más pares de palabras distintas que parecidas.

En la Figura 6.4 se puede ver la relación entre el SMR y DVN, con un coeficiente de correlación de aproximadamente -0.84. De manera similar a gráficos previos, se eligió el uso de box-plots porque ofrecen una descripción resumida de los valores más relevantes. Como es de esperarse en el caso de que la *unicidad* se cumpla, a mayores diferencias entre las palabras, mayor es la distancia de su representación.

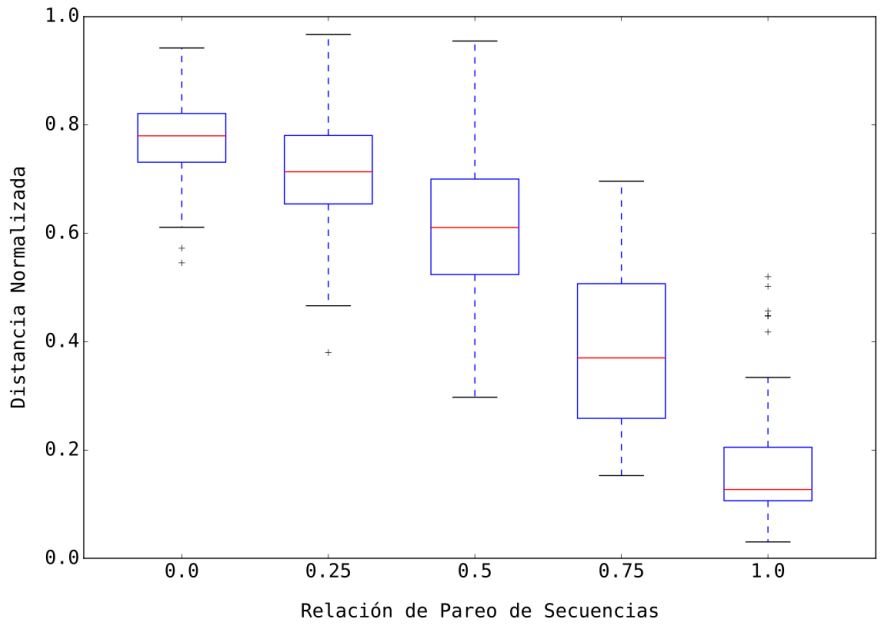

Figura 6.4.: Distancia Normalizada de codificación para pares de palabras en función de la Relación de Pareo de Secuencias.

Teniendo en cuenta estos resultados y que las palabras *conocidas* obtienen representaciones suficientemente distintas como para que su distancia promedio sea superior a la

obtenida después de afectar a estas palabras por errores simples, podemos decir que la propiedad de *unicidad* parece verificarse ya que, no solo se obtienen representaciones distintas, sino que son suficientemente diferentes entre sí para las secuencias válidas.

6.4. Consistencia y Aritmética sobre Representaciones

La *unicidad* es una propiedad necesaria pero no suficiente para obtener el tipo de representación buscada. La clase de codificación que se desea también debería contar con la propiedad de *consistencia*. Para poder verificar que se cumpla esta propiedad será necesario realizar otro tipo de pruebas, en este caso considerando la influencia que tienen sobre la codificación algunas subsecuencias comunes a varias palabras en el conjunto de datos en las formas de, por ejemplo, prefijos, raíces, sufijos, etc.

Figura 6.5.: Representaciones vectoriales para palabras similares a "singing".

Consideremos el caso de la palabra "singing" que intencionalmente no fue incluida en el conjunto de entrenamiento y por lo tanto para el modelo es una palabra *nueva*. En la Figura 6.5 se puede ver como se compara su codificación con la de otras palabras cercanas a su raíz, y otras en la misma categoría gramatical. También podemos considerar de forma similar al caso previo las distancias vectoriales entre estas codificaciones.

$$\begin{aligned} \left\| \vec{sing} - \vec{sings} \right\| &= 14.344 \\ \left\| \vec{sing} - \vec{song} \right\| &= 14.152 \\ \left\| \vec{sings} - \vec{song} \right\| &= 15.059 \end{aligned}$$

$$\begin{aligned} \left\| \vec{sinking} - \vec{singing} \right\| &= 12.912 \\ \left\| \vec{ringing} - \vec{singing} \right\| &= 13.536 \\ \left\| \vec{swinging} - \vec{singing} \right\| &= 15.301 \end{aligned}$$

A pesar de que, como en los casos anteriores, se puede apreciar que las codificaciones para palabras parecidas son similares pero suficientemente distintas entre sí como para diferenciarlas, y que la distancia entre dos palabras similares pero *conocidas* suele ser mayor que la distancia con una palabra *nueva* que está asociada con palabras *conocidas*, lo que queremos verificar ahora específicamente es el grado de influencia de estas

similaridades en la codificación final. Para esto podemos tratar de separar estos elementos y después volver a combinarlos realizando operaciones aritméticas directamente sobre los vectores obtenidos en la codificación. Y más aún, ver si las representaciones obtenidas mediante estas operaciones pueden ser decodificadas en una secuencia válida [78, 116, 91].

El primer ejemplo considerado es el de la palabra "singing" junto a las otras palabras similares ya presentadas. Para este caso se obtuvieron nuevas representaciones mediante operaciones aritméticas reemplazando la raíz en otros gerundios con el verbo "sing", y finalmente se reconstruyeron las secuencias utilizando estas nuevas representaciones. La notación utilizada a continuación indica en **negrita** a la secuencia de patrones correspondientes a la palabra, y las funciones *enc* y *dec* indican correspondientemente la codificación y decodificación de la secuencia hecha por el modelo.

$$\begin{aligned} dec(\ enc(\textbf{sinking}) - enc(\textbf{sink}) + enc(\textbf{sing})\ ) &\rightarrow \textbf{singing} \\ dec(\ enc(\textbf{ringing}) - enc(\textbf{ring}) + enc(\textbf{sing})\ ) &\rightarrow \textbf{singing} \\ dec(\ enc(\textbf{swinging}) - enc(\textbf{swing}) + enc(\textbf{sing})\ ) &\rightarrow \textbf{singing} \end{aligned}$$

El modelo no solo es capaz de reconstruir exitosamente la palabra "singing" a partir de palabras similares, sino que también produce una reconstrucción exitosa aún partiendo de una palabra de distinta longitud, caso que como se había visto, es más difícil desde el punto de vista de los errores tipográficos.

Pero además este mismo principio puede ser aplicado a otras categorías gramaticales también. Volvamos a considerar finalmente algunas de las palabras con menor distancia en la codificación que fueron presentadas al comienzo del capítulo.

$$\begin{aligned} dec(\ enc(\textbf{Victorian}) + enc(\textbf{artists}) - enc(\textbf{artist})\ ) &\rightarrow \textbf{Victorians} \\ dec(\ enc(\textbf{emphatic}) + enc(\textbf{hermetically}) - enc(\textbf{hermetic})\ ) &\rightarrow \textbf{emphatically} \\ dec(\ enc(\textbf{peculiarity}) + enc(\textbf{symmetries}) - enc(\textbf{symmetry})\ ) &\rightarrow \textbf{peculiarities} \end{aligned}$$

Estos ejemplos son especialmente notables dado que, por su menor distancia mutua, hace que las representaciones obtenidas no sean tan claramente diferenciables. Sin embargo, aún entre estas palabras cercanas, es posible reconstruir las secuencias esperadas. Desafortunadamente este no es siempre el caso, y la reconstrucción puede producir secuencias con errores o sencillamente incorrectas.

Para tratar de estudiar los patrones que producen estos errores se recurrió a un conjunto de datos utilizado en uno de los trabajos originales de word2vect [78] que contiene relaciones de palabras del estilo (A→B, C→? [D]), es decir: ¿A es a B como C es a...? D. En donde A, B, y C son palabras de entrada y D debe ser la respuesta. Por ejemplo, (high→higher, narrow→? [narrower]).

Los datos no representan relaciones semánticas de las palabras, sino sintácticas. Para este caso son únicamente relevantes las relaciones morfológicas, con lo cual los casos

irregulares como (good→better, noisy→? [noisier]) empobrecen los resultados, pero fueron igualmente incluidos para utilizar el conjunto completo con sus 8000 relaciones de palabras.

El experimento consistió en codificar las 4 palabras de cada entrada y comparar la Distancia Vectorial Normalizada entre B-A+C y D. Es decir, cuán cercano es el resultado de las operaciones aritméticas al de la palabra buscada.

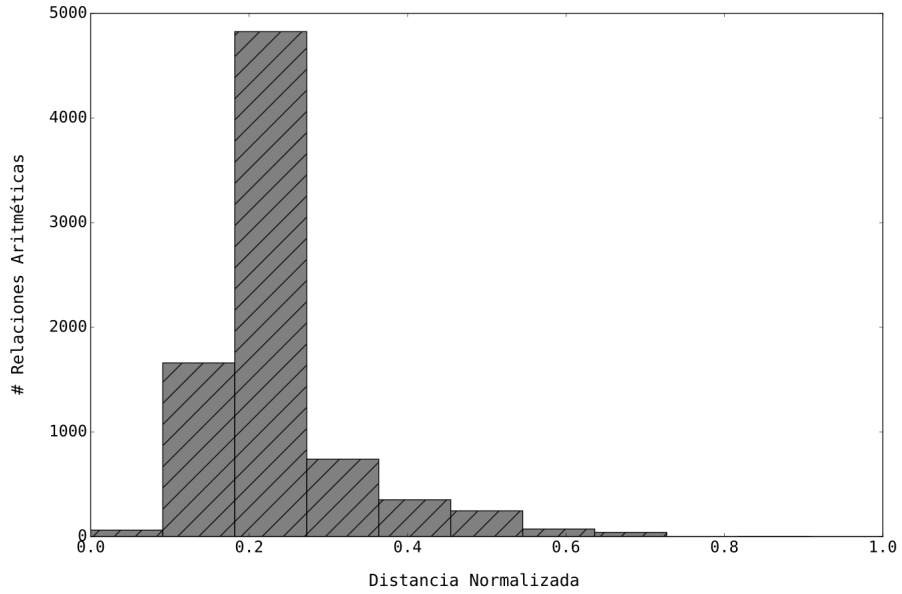

Figura 6.6.: Histograma de la cantidad de relaciones aritméticas sobre la distancia normalizada entre resultados y respuesta.

En la Figura 6.6 se muestra el histograma de la distribución de las distancias entre estas relaciones. Se puede apreciar que las relaciones aritméticas resultantes en distancias cercanas a cero son relativamente pocas, lo cuál explica por qué en la decodificación se suelen producir palabras con errores, diferente de la deseada. Sin embargo, también se puede ver que la distribución de las distancias tiende a producir codificaciones cercanas, lo cual es de esperarse en el caso de que se cumpla la propiedad de *consistencia* [77, 79].

6.5. Conclusiones

En este capítulo se estudió la capacidad del modelo para producir representaciones de secuencias que cumplieran con las propiedades de *unicidad* y *consistencia*, como también la capacidad de estas representaciones para conservar suficiente información como para que sea posible lograr una reconstrucción de la secuencia original, inclusive para secuencias fuera del conjunto de entrenamiento.

En particular se trató con secuencias de letras formando palabras del idioma inglés, y las contingencias relativas a trabajar con información morfológica. De acuerdo a la propiedad de *unicidad* se trató de mostrar que las representaciones de palabras válidas son suficientemente distintas como para poder diferenciarlas aún teniendo en cuenta errores tipográficos simples y que, según la propiedad de *consistencia,* estas representaciones también son regulares y similares para secuencias semejantes.

Pero además se debe destacar que estos resultados no son exclusivos de este dominio, y que sería razonable considerar las posibilidades de obtener resultados similares a nivel semántico trabajando con un lenguaje natural, no solo extrapolando las propiedades aquí demostradas, sino también considerando a este modelo como una potencial primera etapa en el proceso.

Parte de los contenidos y resultados presentados en este capítulo fueron previamente publicados en el artículo "Effective vector representations for variable length symbol sequences" como parte de Computer Science & Information Technology Conference Proceedings vol. 7, no. 6, p. 27-34. (2017)

7. Codificando Información Semántica con Lenguajes Naturales

7.1. Introducción

La interpretación de sentencias expresadas en un lenguaje natural es uno de los mayores retos en inteligencia artificial. Aún estando limitado a un dominio específico la gran diversidad en sutiles variaciones que pueden ser usadas para expresar conceptos semánticamente equivalentes o muy similares es enorme, potencialmente infinita. Además esta misma diversidad hace más difícil juzgar la equivalencia semántica entre distintas sentencias, convirtiéndolo en algo más subjetivo, en donde un análisis puntual, sentencia por sentencia, podría no ser el enfoque más adecuando para el problema [4, 2].

Aunque hacer tal evaluación es trivial para las personas, la construcción de algoritmos y modelos computacionales que imiten el desempeño a nivel humano representa un problema de comprensión del lenguaje natural difícil y profundo [17]. En este capítulo se estudiará la capacidad del modelo propuesto, considerando sentencias como secuencias de palabras, y palabras como secuencias de letras, para obtener codificaciones que representen información semántica.

7.2. Similitud Textual Semántica

Existe un tipo de prueba especialmente diseñada para evaluar la capacidad de un modelo para realizar interpretaciones semánticas de sentencias expresadas en un lenguaje natural. Esta prueba es la llamada Similitud Textual Semántica[1] y consiste en medir el grado de equivalencia semántica subyacente entre pares de fragmentos de texto. La tarea consiste en, dadas dos sentencias, devolver una puntuación de similitud continua en una escala de 0 a 5, donde 0 indica que la semántica de las oraciones es completamente independiente y 5 significa equivalencia semántica.

Existen varios conjuntos de datos especialmente preparados para esta tarea y que pueden ser usados para el entrenamiento y la evaluación del modelo. Las instancias de estos datos

[1] En inglés: Semantic Textual Similarity.

comprenden pares de sentencias en el idioma inglés que provienen de descripciones de imágenes, definiciones de palabras, titulares de noticias, etc. y la puntuación de similitud que fue asignada por jueces humanos. De esta forma es de esperarse que, por ejemplo, descripciones de la misma imagen hechas por distintas personas, o titulares de la misma noticia provenientes de distintas fuentes, tiendan a tener una puntuación cercana a 5.

Los datos elegidos para la experimentación pertenecen a la competencia SemEval-2017 en las tareas de Semantic Textual Similarity (STS) para pares de sentencias en inglés [11]. Esta tarea fue elegida para la experimentación porque, en relación a otras tareas similares como Sentiment Analisys o Recognizing Textual Entailment (RTE), parece la más adecuada para probar la capacidad del modelo en capturar información semántica. La competencia SemEval-2017 fue elegida por ser la más reciente para la que existen publicados abiertamente los datos y resultados involucrados de todos los participantes. Si bien la competencia SemEval tiene ediciones más recientes, hasta el 2021 inclusive, las tareas involucradas en competencias posteriores fueron más diversificadas y no se volvió a incluir la de STS de forma independiente.

Los datos provistos incluyen tres conjuntos que pueden ser utilizados en entrenamiento (train), validación (dev) y evaluación (test). En total conforman unas 8600 entradas, provenientes aproximadamente en un 50 % de noticias, 35 % descripción de imágenes, y 15 % de respuestas en foros. Cada una de estas entradas incluye un par de sentencias en inglés con la correspondiente puntuación de similitud semántica según evaluaciones hechas por jueces humanos.

Una mínima cantidad de sentencias fueron excluidas del conjunto de datos debido a que, por su longitud en palabras, no hubieran podido ser procesadas adecuadamente por la arquitectura elegida.

7.3. Codificación de Palabras y Sentencias

Aunque, en principio, sería posible tratar a cada sentencia directamente como una secuencia de letras, números, espacios, y potencialmente otros símbolos, esto requeriría de un modelo con una cantidad innecesariamente grande de etapas. En este caso se adoptó una solución más práctica, aprovechando la noción de *palabra,* para poder realizar la codificación en dos bloques. El primero encargado de codificar palabras en vectores, y el segundo de codificar sentencias, tomando como entrada las codificaciones de palabras producidas por el bloque anterior. Es posible que esta estrategia no sea la más adecuada con otros idiomas, pero para el castellano, el inglés, y la mayoría de los lenguajes occidentales es completamente válida.

Para el entrenamiento del primer bloque se utilizó un conjunto de 20 mil palabras elegidas al azar provenientes de un diccionario genérico de palabras en inglés de unas 70 mil entradas. Las palabras fueron consideradas como secuencias de letras y números, utilizando como patrón de entrada para representar cada símbolo la codificación binaria de dimensión 8 correspondiente valor ASCII.

Para el entrenamiento del bloque de sentencias se utilizó un vuelco de datos proveniente de la versión en Inglés Simplificado de Wikipedia [21]. Este conjunto de datos comprende unos 50 mil artículos en texto plano que se corresponden con unas 375 mil sentencias [github.com/lgdoor/dump-of-simple-english-wiki]. De estas sentencias fueron elegidas 100 mil al azar para el conjunto de entrenamiento.

Al utilizar datos de orígenes disimiles es posibles apreciar la capacidad del modelo para ser trasladado a distintos dominios.

Para la representación de las secuencias de entrada es posible adoptar distintas estrategias. Hasta ahora la estrategia utilizada consistió en usar un símbolo distinguido para indicar el punto de finalización y procesar la secuencia sólo hasta esta posición. Sin embargo para estas pruebas se utilizó un método ligeramente distinto. En lugar de un símbolo distinguido al final se utilizó uno al comienzo de la secuencia, y las secuencias se repitieron hasta alcanza la longitud máxima posible en cada bloque. Esto fue útil para introducir un nivel de redundancia que permita aprovechar al máximo la capacidad de los modelos.

Todas las sentencias fueron preprocesadas para remover signos de puntuación, símbolos, y caracteres inválidos, de forma que las sentencias fueran secuencias de palabras separadas por espacios, y las palabras secuencias de únicamente letras y números. En la Tabla 7.1 pueden verse algunos resultados provenientes de los datos de testeo, antes y después del preprocesado, para varias sentencias.

El entrenamiento se pudo realizar de forma progresiva dentro de cada bloque, etapa por etapa. Esto permitió determinar la arquitectura experimentalmente, basándose en una estimación inicial de la dimensión de cada capa a partir de la cantidad de patrones,

```
If you want to buy $ billion in XYZ Inc. common stock, who cares?
if you want to buy billion in xyz inc common stock who cares

Hi Eots: YOU FOOLS..bought the cover-up..hook..
hi eots you fools bought the cover up hook

Umm, that pretty clearly states, "the statute is presumed constitutional."
umm that pretty clearly states the statute is presumed constitutional

naeem m. smuggled 10 kilograms (22 pounds) of heroin.
naeem m smuggled 10 kilograms 22 pounds of heroin

#US details help in failed #hostage rescue.
us details help in failed hostage rescue
```

Cuadro 7.1.: Ejemplos del pre-procesado de las sentencias para eliminar símbolos menos relevantes.

unidades y convergencia al entrenar la etapa anterior, sin tener que recurrir a una exploración exhaustiva. Por cada etapa fueron entrenadas al menos dos redes neuronales de distintas dimensiones además de la estimada inicialmente. En todos los casos se favoreció la arquitectura que ofreciera mejores resultados con la menor cantidad de parámetros entrenables. Como el objetivo final comprendía únicamente obtener una representación vectorial sin mantener suficiente información para reconstruir la secuencia original, fue posible realizar una mayor reducción de las dimensiones, tanto en el bloque de palabras, como en el bloque de sentencias.

La arquitectura para el bloque de palabras entonces quedó conformada finalmente por 7 etapas de 13, 25, 41, 50, 57, 65, y 76 unidades de salida, con pasos 1, 1, 1, 1, 2, 2, y 2 correspondientemente. Mientras que el bloque de sentencias fue formado por 8 etapas de 91, 109, 130, 156, 187, 224, 268, y 321 unidades de salida, con pasos 1, 1, 1, 2, 2, 2, 2, y 2 respectivamente, que le permiten procesar efectivamente oraciones de hasta 35 palabras.

Una vez que el bloque de palabras fue entrenado se lo utilizó para codificar todas las palabras que componían cada una de las sentencias del conjunto de datos de entrenamiento del segundo bloque. Con estas secuencias de palabras codificadas el bloque de sentencias fue entrenado para, finalmente, poder codificar sentencias a vectores de dimensión fija.

7.4. Experimentación y Resultados

Dentro de las tareas de prueba de evaluación semántica SemEval existen distintas categorías que permiten una comparación de resultados más adecuada. Las principales distinciones provienen del grado de supervisión para el entrenamiento y del tipo de dependencia entre pares de sentencias para obtener su codificación.

El grado de supervisión se refiere al tipo de datos que fueron usados para el entrenamiento. La competencia ofrece 3 conjuntos de datos para entrenamiento, validación y evaluación. Las categorías consisten en:

no-supervisado: no se usan los datos de entrenamiento o validación provistos.

semi-supervisado: solo se usan datos de validación (por ejemplo para el ajuste o elección de parámetros).

supervisado: solo se usan datos de entrenamiento y validación.

sin-restricciones: además de los datos de entrenamiento y validación se usan otros datos similares (por ejemplo de competencias anteriores).

El grado de relación en la codificación de las sentencias se refiere al uso de algún tipo de interacción durante el entrenamiento entre los pares de sentencias provistas (por ejemplo mecanismos de atención, alineamiento, superposición, etc). Las categorías consisten en:

independiente: la representación de las sentencias es computada independientemente.

dependiente: existe algún tipo de interacción entre las sentencias para obtener su representación.

La evaluación que se llevó a cabo consistió en, para cada instancia del conjunto de datos de testeo provisto para la competencia SemEval-2017, codificar el par de sentencias utilizando los modelos entrenados, y comparar las diferencias entre los dos vectores resultantes con la puntuación de similitud semántica dada como atributo. Bajo estas condiciones, como el modelo fue entrenado utilizando sentencias provenientes de la versión en Inglés Simplificado de Wikipedia, se corresponde con la categoría *no-supervisado*. Y como la codificación de cada sentencia se realizó de forma individual las pruebas corresponden a la categoría *independiente*.

Para calcular la diferencia entre codificaciones de dos sentencias se recurrió a la *similitud del coseno (cossim)* que puede ser expresada como el producto interno normalizado entre dos vectores ($^{u \cdot v}/_{\|u\| \times \|v\|}$). Esta medida indica qué tan ortogonales son los dos vectores evaluados, tomando valores que tienden a 1 para direcciones similares, -1 para orientaciones similares pero direcciones opuestas, y 0 para direcciones ortogonales. A pesar de que esta medida es similar a la norma de la diferencia de los vectores ($\|u - v\|$), estimación propuesta previamente, se prefirió utilizar esta otra, no solo para mostrar el comportamiento del modelo utilizando una forma de medida alternativa, sino por ser más estándar en este tipo de pruebas.

Para la evaluación final se utilizó la similitud del coseno comparada en función de la puntuación de similitud semántica para cada par de sentencias provistas por los datos de

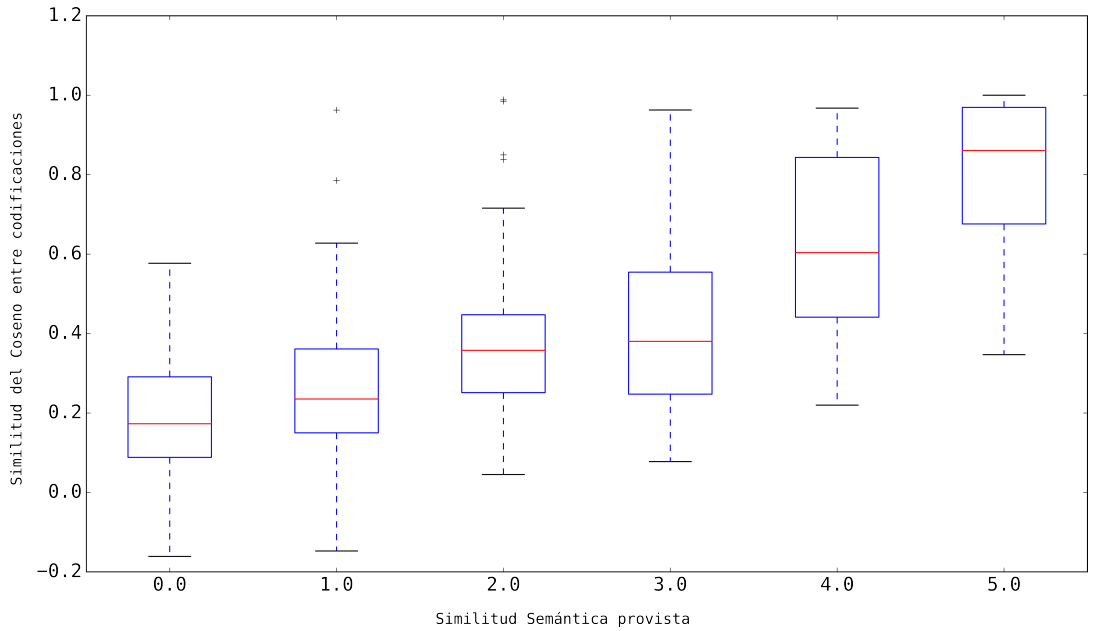

Figura 7.1.: Ortogonalidad entre las representaciones de pares de sentencias en función de la puntuación de similitud semántica propuesta por jueces humanos. Se muestra que las sentencias menos relacionadas tienden a ser ortogonales.

0.2	0.039	two men standing in grass staring at a car
		a woman in a pink top posing with beer
1.2	0.219	ukraine protest leaders name ministers russian troops on alert
		ukraine refuses to act against russian provocation
2.4	0.468	two women sitting outside laughing
		two women standing outside of a bus
3.0	0.807	you can use it too
		you can do it too
4.4	0.781	a yellow car speeds along a snowy field
		a yellow car drives quickly in the snow

Cuadro 7.2.: Ejemplo de comparaciones entre las puntuaciones de similitud semántica provistas (primera columna), la similitud del coseno entre codificaciones (segunda columna) y los pares de sentencias originales (tercera columna).

testeo. En la Figura 7.1 se puede apreciar la relación, correspondiente con un coeficiente de correlación de Pearson de 0.597, entre las dos medidas. Vale la pena señalar que las sentencias con una menor similitud semántica tienen una tendencia de *cossim* a 0 según la evaluación, es decir que dentro del espacio semántico al que las sentencias son proyectadas, los significados más distintos tienden a ser ortogonales [44, 3, 114].

En la Tabla 7.2 también se pueden ver algunos ejemplos de estas medidas para pares de sentencias pertenecientes al conjunto de datos de testeo.

Finalmente, para poder apreciar estos resultados con una perspectiva más general, en la Tabla 7.3 se ofrece una comparación con algunos de los otros modelos participantes en la misma competencia dentro de la misma categoría. Puede observarse que los resultados aquí obtenidos calificarían aproximadamente con una relación de eficacia media con respecto a los otros resultados publicados, pero debe tenerse en cuenta que en general las arquitecturas participantes tienen una cantidad de parámetros entrenables en el orden de millones [nlp.stanford.edu/projects/snli] mientras que los modelos utilizados en esta

Modelo	**Puntaje**
Glove [88]	40.6
...	
Doc2Vec [29]	59.2
Modelo propuesto	*59.7*
C-PHRASE [90]	63.9
...	
Unsup. SIF on ParaNMT [32]	79.5

Cuadro 7.3.: Comparación de resultados con otros modelos participantes en SemEval-2017. El puntaje corresponde al coeficiente de correlación de Pearson.

evaluación final suman apenas unos 270 mil parámetros entrenables entre los bloques de palabras y sentencias.

7.5. Conclusiones

Para poder estudiar la capacidad del método propuesto en la codificación de sentencias en inglés a un espacio de representación semántico se utilizaron dos modelos. Un primero un modelo fue utilizado para codificar palabras como secuencias de letras y, usando estas representaciones de palabras, un segundo modelo para codificar sentencias como secuencias de palabras. Entre estas codificaciones producidas para pares de sentencias es posible observar una correlación positiva con la percepción de similitud semántica estimada por seres humanos.

Esto demuestra que el modelo es capaz de capturar información semántica proveniente de lenguajes naturales, aunque los resultados obtenidos no sean comparables a los mejores posibles demostrados para esta tarea. Sin embargo, el modelo propuesto cuenta con la ventaja de no estar limitado a un vocabulario fijo ni modelos adicionales preentrenados, puede ser entrenado de forma completamente auto-supervisada, y está en un orden de magnitud por debajo en la cantidad de parámetros entrenables con respecto a otros modelos utilizados en tareas similares.

Conclusiones Generales

Originalmente se planteó la necesidad de crear un modelo que pudiera codificar información presentada en forma secuencial a una representación vectorial de dimensión fija. Además, si la fuente de esta información provenía de un lenguaje natural, esta representación debería tener la capacidad de capturar información semántica.

El modelo propuesto en este trabajo no solo muestra que con la adopción de hipótesis provenientes de la semántica cognitiva es posible cumplir con estos objetivos, sino que también pueden ser evitados inconvenientes aún presentes en algunas técnicas contemporáneas, como ser dependiente de un vocabulario o una gramática en particular y, gracias a las representaciones vectoriales de sentencias que permiten la integración con otras estrategias de aprendizaje automático, tampoco estar ligado a un tipo de tarea específica.

Uno de los aspectos fundamentales para poder obtener estos resultados fue la organización del modelo en etapas, que no solo permite un entrenamiento progresivo sino que, al codificar las regularidades propias de las combinaciones de pares de patrones independientemente en cada nivel, le permite adaptarse a distintos tipos de problemas y brinda una mayor flexibilidad para la elección de la arquitectura manteniendo una cantidad relativamente baja de parámetros entrenables (Capítulo 2).

Sin embargo, el tipo de representación dispersa producida por la codificación en las distintas etapas tiende a formar problemas de mayor dificultad para las estrategias de descenso por gradiente típicas. Por esto se propuso un nuevo tipo de algoritmo de entrenamiento que, en su naturaleza estocástica, tiende a que sea más probable que el costo disminuya. Esta estrategia no solo permitió disminuir los tiempos de entrenamiento sino que además mostró ser efectiva con otras variedades de problemas (Capítulos 3 y 4).

Para poder comprobar todas estas propiedades del modelo propuesto fueron llevadas a cabo varias instancias de experimentación.

Primero, al establecer una noción de *significado* a partir de la Hipótesis Distribucional y la Teoría de Prototipos, fue posible observar la capacidad para extraer información semántica para un conjunto de sentencias generadas a partir de una gramática libre de contexto. El uso de una gramática de este tipo permitió tener un mayor control sobre las variaciones y casos posibles utilizados durante el entrenamiento y la posterior evaluación (Capítulo 5).

Después, la comparación de las relaciones entre pares de palabras distintas según la proporción de secuencias más largas en común, y la distancia vectorial normalizada para

las codificaciones producidas por el modelo, parece sugerir que se preserva la propiedad de *unicidad*. Así como también, las distancias normalizadas resultantes de operaciones aritméticas sobre vectores que representan a dos pares de palabras sintácticamente relacionadas, tienden a indicar un comportamiento como el esperado según la propiedad de *consistencia*. Adicionalmente se estudió la robustez del modelo ante errores tipográficos simples y la existencia de palabras fuera del vocabulario (Capítulo 6).

Finalmente, en la representación de sentencias pertenecientes a un lenguaje natural, se pudo observar una correlación entre la similitud de las codificaciones y la equivalencia semántica estimada por humanos para pares de sentencias provenientes de distintas fuentes. La representación de las sentencias fue conseguida mediante dos modelos, uno para codificar secuencias de letras en palabras, y otro para codificar secuencias de palabras en sentencias. Para ambos modelos los datos de entrenamiento provinieron de orígenes distintos a los utilizados finalmente en las pruebas de evaluación (Capítulo 7).

Sin embargo la llegada a estos resultados no ocurrió sin inconvenientes.

Una de las principales barreras para poder utilizar más amplia y fácilmente el modelo propuesto es el elevado número de pruebas experimentales necesarias para encontrar la arquitectura óptima en cada etapa. Esto surge de la necesidad de buscar un equilibrio entre la menor dimensión de representación posible y suficientes parámetros entrenables para un aprendizaje adecuado.

Otra de las potenciales limitaciones para una mayor adopción de la propuesta es que la longitud máxima de secuencia válida está directamente relacionada con la cantidad de etapas. Lo cual, en conjunción con el punto anterior, dificultaría su uso en otros dominios como el procesamiento de sonido, en donde las secuencias de datos son intrínsecamente más largas.

Además de estas consideraciones generales, durante la experimentación también surgieron otras dificultades más específicas.

A pesar de que uno de los objetivos del modelo es no depender directamente del uso de word-embeddings, poder contar con codificaciones vectoriales de palabras individuales en un espacio de representación semántica puede ser evidentemente muy útil, pero la adaptación de la propuesta para esta tarea no produjo resultados del todo satisfactorios. Sin embargo esto no significa que su desempeño no pueda mejorarse, por ejemplo utilizando directamente otra forma alternativa de representación en la entrada.

De manera similar, la tarea de corrección o detección de errores tipográficos parece entrar en conflicto con la capacidad para trabajar con palabras fuera de vocabulario. Esto es debido al sutil equilibrio que debe existir para que el modelo pueda funcionar como un tipo de memoria auto-asociativa y la capacidad para generalizar a nuevos datos. Aunque no definitiva, una potencial y sencilla solución a esto sería entrenar dos modelos específicos por separado para cada una de estas tareas.

Finalmente, la misma naturaleza de la arquitectura formada exclusivamente por autocodificadores apilados parecía prometer un mecanismo para la generación de secuencias

sin costo adicional, producto del entrenamiento normal del modelo. Sin embargo, los resultados exitosos en la reconstrucción de secuencias a partir de codificaciones vectoriales fueron limitados y merecen más investigación.

Por otro lado, algunas de las técnicas e ideas aquí propuestas tienen el potencial de ser aprovechadas en otros dominios que exceden ampliamente el de este trabajo. Por ejemplo, el método de codificación dispersa parece especialmente adecuado para su uso en el procesamiento de imágenes digitales, en donde para distintos filtros o características visuales existen diferentes grupos de correlaciones entre variables. De manera similar, la técnica propuesta de descenso por gradiente, podría en principio no solo ser utilizada en cualquier otro dominio en donde se utilizan técnicas similares, sino también en otros sujetos a restricciones adicionales, como una reciente colaboración para el entrenamiento de memristores parece sugerir. Así mismo, la idea de utilizar una Gramática Libre de Contexto para generar conjuntos de datos con características deseadas específicas que puedan ser utilizados a modo de ground-truth puede ser aprovechada y adaptada para muchos dominios de problemas en donde los datos de buena calidad y/o correctamente etiquetados son costosos o escasos.

Pero fundamentalmente, quizás el mayor aporte de este trabajo sea la propuesta del modelo en sí misma, al estar inspirada por principios derivados de una teoría lingüística como la semántica cognitiva. No sólo porque brinda un enfoque original y técnicas novedosas en el área del Procesamiento del Lenguaje Natural, en donde no existen muchas perspectivas alternativas, sino porque también ofrece resultados similares a partir de principios distintos de los utilizados ampliamente en la actualidad. Aún los modelos basados en atención, como las variantes de los Transformers, se basan en un *modelado del lenguaje* a partir de la predicción de la siguiente palabra en una sentencia. Si bien esta técnica a demostrado ser indiscutiblemente efectiva, es en principio más probabilística estadística que lingüística, y casi exclusivamente la única utilizada actualmente.

Un modelo basado en principios distintos puede ofrecer ventajas distintas a las de los modelos state-of-the-art. Por ejemplo, la organización en etapas y la menor cantidad de parámetros permite un entrenamiento progresivo y que requiere menor poder de cómputo, haciéndolo más accesible. Y la codificación de secuencias en representaciones vectoriales permitiría su integración con otros métodos de aprendizaje automático más fácilmente que a partir de otro tipo de modelos.

Si bien los resultados obtenidos no fueron comparables a los mejores existentes, esta propuesta cumple con el fin de presentarse como una prueba-de-concepto con el potencial de obtener mejores y más variados resultados en futuras investigaciones.

Por ejemplo, en la función de activación para la codificación dispersa sería posible mejorar los resultados mediante el uso de estimadores diferentes o con otro tipo de función logística. De manera similar, sería conveniente tratar de formalizar una demostración de convergencia para el Gradient Omissive Descent y a partir de esta ver si es posible refinar el algoritmo. En ambos casos, sería además muy interesante probar su eficacia con un conjunto más amplio y variado de problemas.

Otras áreas en las que también se beneficiaría ampliamente la investigación serían la

posibilidad de encontrar algún tipo de estimador para facilitar el dimensionamiento de la arquitectura del modelo, aunque sea progresivamente durante el entrenamiento, y la posibilidad de incorporar algún tipo de codificación recursiva, similar al mecanismo utilizado por las redes RAAM, que permita la codificación de secuencias de mayor longitud.

Pero el prospecto más atractivo está definitivamente en la posibilidad de probar al modelo propuesto no solo en otras tareas asociadas al Procesamiento de Lenguajes Naturales, sino también con otro tipo problemas secuenciales, finalmente aprovechando la escalabilidad de la arquitectura para lograr resultados más competitivos.

Agradecimientos

A las personas cercanas que siguieron mis progresos
y supieron cuando intervenir en mis procesos cognitivos.

A. Especificación de la Gramática

S → Faccion | Fvigilia | Fsimul

Faccion → FVa | FVa IL | IL FVa

Fvigilia → Personaje Vvigilia FVa

Fsimul → FVa Vsimul FVa

FVa → FVpersigue | FVgolpea

FVpersigue → Perro Persigue Gato | Gato Persigue Raton | Gato Persigue Canario | Perro Persigue Cartero | Raton Persigue Señora | Gato EsPerseg Perro | Raton EsPerseg Gato | Canario EsPerseg Gato | Cartero EsPerseg Perro | Señora EsPerseg Raton

Persigue → *persigue a* | *corre detras de*

EsPerseg → *es perseguido por* | *corre delante de*

FVgolpea → Agresor Golpea Gato | Gato Golpea Victima | Gato EsGolp Agresor | Victima EsGolp Gato

Golpea → *golpea a* | *le pega a*

EsGolp → *es golpeado por* | *recibe un golpe de*

Agresor → Perro | Raton | Señora

Victima → Raton | Canario | Humano

Humano → Señora | Cartero

Animal → Perro | Gato | Raton | Canario

Perro → ArtM *perro* | *butch* | *spike*

Gato → ArtM *gato* | *tom* | *silvestre*

Raton → ArtM *raton* | *jerry*

Canario → ArtM *canario* | *tweety*

Señora → *la señora*

Cartero → *el cartero*

ArtM → *el* | *un*

ArtF → *la* | *una*

IL → Lugar | *con* Instrumento | *con* Icocina Lcocina | Lcocina *con* Icocina | *con* Isala Lsala | Lsala *con* Isala | *con* Ipatio Lpatio | Lpatio *con* Ipatio | *con* Icalle Lcalle | Lcalle *con* Icalle

Lugar → Lcasa | Lcalle

Lcasa → PLE *la casa* | Lcocina | Lsala | Lpatio

Lcalle → PLE *la calle* | Pcalle

Lcocina → PLE *la cocina* | Pcocina

Lsala → PLE *la sala* | Psala

Lpatio → PLE *el patio* | Ppatio

Pcocina → PLG *la heladera* | PLG *la mesa* | PLG *la ventana*

Psala → PLG *el sillon* | PLG *la chimenea* | PLG *la puerta*

Ppatio → PLG *el jardin* | PLG *el arbol* | PLG *la cucha*

Pcalle → PLG *el tacho* | PLG *el poste*

PLE → *en* | *dentro de* | PLG

PLG → *por* | *junto a* | *delante de*

Instrumento → Icasa | Icalle

Icasa → Icocina | Isala | Ipatio

Icocina → *la escoba* | *la botella* | *la sarten* | ArtM *jarron*

Isala → *la escoba* | *el florero* | *el atizador*

Ipatio → *la escoba* | *el ladrillo* | *el martillo* | ArtF *pala*

Icalle → *la botella* | *el ladrillo* | *el fierro* | ArtM *palo*

Fvigilia → Personaje Vvigilia FVa

Vvigilia → *observa como* | *mira como* | *sabe que* | *sospecha que*

Personaje → Humano | Animal

Fsimul → FVa Vsimul FVa

Vsimul → *mientras que* | *cuando*

Bibliografía

[1] L F Abbott, *Learning in neural network memories*, Network: Computation in Neural Systems **1** (1990), no. 1, 105–122.

[2] James F. Allen, *Natural language understanding (2. ed.)*, Benjamin/Cummings, 1994.

[3] James F. Allen, Mary Swift, and Will de Beaumont, *Deep semantic analysis of text*, Proceedings of the 2008 Conference on Semantics in Text Processing (Stroudsburg, PA, USA), STEP '08, Association for Computational Linguistics, 2008, pp. 343–354.

[4] Madeleine Bates, *Models of natural language understanding*, Voice Communication Between Humans and Machines (David B. Roe and Jay G. Wilpon, eds.), The National Academies Press, 1994, pp. 238–253.

[5] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent, *A neural probabilistic language model*, Advances in Neural Information Processing Systems **13** (2000).

[6] Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle, *Greedy layer-wise training of deep networks*, In NIPS, MIT Press, 2007.

[7] David M Blei, Andrew Y Ng, and Michael I Jordan, *Latent dirichlet allocation*, Journal of machine Learning research **3** (2003), no. Jan, 993–1022.

[8] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov, *Enriching word vectors with subword information*, Transactions of the association for computational linguistics **5** (2017), 135–146.

[9] John S. Bridle, *Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition*, Neurocomputing (Berlin, Heidelberg) (Françoise Fogelman Soulié and Jeanny Hérault, eds.), Springer Berlin Heidelberg, 1990, pp. 227–236.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al., *Language models are few-shot learners*, Advances in neural information processing systems **33** (2020), 1877–1901.

[11] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, *SemEval-2017 Task 1: Semantic Textual Similarity - Multilingual and Cross-lingual Focused Evaluation*, ArXiv e-prints (2017).

[12] Daniel Cer, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, et al., *Universal sentence encoder*, arXiv preprint arXiv:1803.11175 (2018).

[13] Danqi Chen and Christopher Manning, *A fast and accurate dependency parser using neural networks*, Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 740–750.

[14] N. Chomsky, *Syntactic structures*, Janua linguarum: Series minor, Mouton, 1957.

[15] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun, *The loss surfaces of multilayer networks*, Artificial Intelligence and Statistics, 2015, pp. 192–204.

[16] Adam Coates and Andrew Y Ng, *The importance of encoding versus training with sparse coding and vector quantization*, Proceedings of the 28th international conference on machine learning (ICML-11), 2011, pp. 921–928.

[17] Ronan Collobert and Jason Weston, *A unified architecture for natural language processing: deep neural networks with multitask learning.*, ICML (William W. Cohen, Andrew McCallum, and Sam T. Roweis, eds.), ACM International Conference Proceeding Series, vol. 307, ACM, 2008, pp. 160–167.

[18] Pierre Comon, *Independent component analysis, a new concept?*, Signal processing **36** (1994), no. 3, 287–314.

[19] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes, *Supervised learning of universal sentence representations from natural language inference data*, arXiv preprint arXiv:1705.02364 (2017).

[20] Alexis Conneau, Holger Schwenk, Loïc Barrault, and Yann Lecun, *Very deep convolutional networks for text classification*, arXiv preprint arXiv:1606.01781 (2016).

[21] William Coster and David Kauchak, *Simple english wikipedia: a new text simplification task*, Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, 2011, pp. 665–669.

[22] Christian Darken, Joseph Chang, and John Moody, *Learning rate schedules for faster stochastic gradient search*, Neural Networks for Signal Processing [1992] II., Proceedings of the 1992 IEEE-SP Workshop, 1992, pp. 3–12.

[23] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio, *Identifying and attacking the saddle point problem in high-dimensional non-convex optimization*, Advances in neural information processing systems, 2014, pp. 2933–2941.

[24] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, arXiv preprint arXiv:1810.04805 (2018).

[25] Rahul Dey and Fathi M Salem, *Gate-variants of gated recurrent unit (gru) neural networks*, 2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS), IEEE, 2017, pp. 1597–1600.

[26] Thomas G Dietterich, *Machine learning for sequential data: A review*, Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), Springer, 2002, pp. 15–30.

[27] David L. Donoho and Michael Elad, *Optimally sparse representation in general (nonorthogonal) dictionaries via l1 minimization*, Proceedings of the National Academy of Sciences **100** (2003), no. 5, 2197–2202.

[28] Timothy Dozat, *Incorporating nesterov momentum into adam*, Natural Hazards **3** (2016), no. 2, 437–453.

[29] Mirela-Stefania Duma and Wolfgang Menzel, *Unsupervised knowledge-free semantic textual similarity via paragraph vector*, Proceedings of the 11th international workshop on semantic evaluation (SEF@UHH at SemEval-2017), 2017, pp. 170–174.

[30] Susan T Dumais, George W Furnas, Thomas K Landauer, Scott Deerwester, and Richard Harshman, *Using latent semantic analysis to improve access to textual information*, Proceedings of the SIGCHI conference on Human factors in computing systems, 1988, pp. 281–285.

[31] Jeffrey L. Elman, *Finding structure in time*, Cognitive Science **14** (1990), no. 2, 179–211.

[32] Kawin Ethayarajh, *Unsupervised random walk sentence embeddings: A strong but simple baseline*, Proceedings of The Third Workshop on Representation Learning for NLP, 2018, pp. 91–100.

[33] Vyvyan Evans, Benjamin Bergen, and Jorg Zinken, *The cognitive linguistics enterprise: An overview*, The Cognitive Linguistics Reader (2006).

[34] Gottlob Frege, *Sense and Reference*, Philosophical Review **57** (1948), no. 3, 209–230.

[35] Peter Gärdenfors, *Meanings as conceptual structures*, Lund University Cognitive Studies (1995).

[36] Dirk Geeraerts, *The theoretical and descriptive development of lexical semantics*, The lexicon in focus. Competition and convergence in current lexicology (2002), 23–42.

[37] Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya, *Multilingual language processing from bytes*, arXiv preprint arXiv:1512.00103 (2016).

[38] Marco Gori and Alberto Tesi, *On the problem of local minima in backpropagation*, IEEE Transactions on Pattern Analysis and Machine Intelligence **14** (1992), no. 1, 76–86.

[39] Karol Gregor and Yann LeCun, *Learning fast approximations of sparse coding*, ICML, 2010, pp. 399–406.

[40] Gustavo A. Lado and Enrique C. Segura, *Gradient omissive descent is a minimization algorithm*, International Journal on Soft Computing, Artificial Intelligence and Applications **8** (2019), no. 1, 37–45.

[41] Gustavo Lado and Enrique Carlos Segura, *Effective vector representations for variable length symbol sequences*, Computer Science & Information Technology Conference Proceedings **7** (2017), no. 6, 27–34.

[42] T Harada, O Araki, and A Sakurai, *Learning context-free grammars with recurrent neural networks*, IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222), 2602–2607.

[43] Zellig S Harris, *Distributional structure*, Word **10** (1954), no. 2-3, 146–162.

[44] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom, *Teaching machines to read and comprehend*, Advances in Neural Information Processing Systems 28 (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), Curran Associates, Inc., 2015, pp. 1693–1701.

[45] Geoffrey Hinton and Ruslan Salakhutdinov, *Discovering binary codes for documents by learning deep generative models.*, Top Cogn Sci **3** (2011), no. 1, 74–91.

[46] Geoffrey E Hinton, *Learning multiple layers of representation.*, Trends Cogn. Sci. (Regul. Ed.) **11** (2007), no. 10, 428–34.

[47] Geoffrey E. Hinton and Ruslan R. Salakhutdinov, *Reducing the dimensionality of data with neural networks*, science **313** (2006), no. 5786, 504–507.

[48] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), no. 8, 1735–1780.

[49] Timo Honkela, *Self-organizing maps of words for natural language processing applications*, Proceedings of the International ICSC Symposium on Soft Computing, Citeseer, 1997, pp. 401–407.

[50] Vayrynen Honkela, Hyvarinen, *WordICA—Emergence of linguistic representations for words by independent component analysis*, (2010).

[51] J. J. Hopfield, *Neurocomputing: Foundations of research*, MIT Press, Cambridge, MA, USA, 1988, pp. 457–464.

[52] Aapo Hyvärinen, *Independent component analysis: recent advances*, Phil. Trans. R. Soc. A **371** (2013), no. 1984, 20110534.

[53] Aapo Hyvärinen and Erkki Oja, *Independent component analysis by general nonlinear hebbian-like learning rules*, Signal Processing **64** (1998), no. 3, 301–313.

[54] Mohit Iyyer, Varun Manjunatha, Jordan Boyd-Graber, and Hal Daumé III, *Deep unordered composition rivals syntactic methods for text classification*, Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers), 2015, pp. 1681–1691.

[55] J Karhunen, *Nonlinear independent component analysis*, ICA: Principles and Practice (2001), 113–134.

[56] Juha Karhunen, Erkki Oja, Liuyue Wang, Ricardo Vigario, and Jyrki Joutsensalo, *A class of neural networks for independent component analysis*, IEEE Transactions on neural networks **8** (1997), no. 3, 486–504.

[57] Samuel Kaski and Teuvo Kohonen, *Winner-take-all networks for physiological models of competitive learning*, Neural Networks **7** (1994), no. 6, 973 – 984, Models of Neurodynamics and Behavior.

[58] Yoon Kim, Yacine Jernite, David Sontag, and Alexander M. Rush, *Character-aware neural language models*, Thirtieth AAAI conference on artificial intelligence (2016).

[59] Scott Kirkpatrick, C. Daniel Gelatt, Mario P. Vecchi, et al., *Optimization by simulated annealing*, science **220** (1983), no. 4598, 671–680.

[60] Ryan Kiros, Yukun Zhu, Russ R Salakhutdinov, Richard Zemel, Raquel Urtasun, Antonio Torralba, and Sanja Fidler, *Skip-thought vectors*, Advances in neural information processing systems **28** (2015).

[61] Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya, *Reformer: The efficient transformer*, arXiv preprint arXiv:2001.04451 (2020).

[62] Gustavo A Lado and Enrique C Segura, *Coding with Logistic Softmax Sparse Units*, AIFU 2020 : 6th International Conference on Artificial Intelligence and Applications (2020).

[63] George Lakoff, *Women, fire and dangerous things: What categories reveal about the mind*, University of Chicago Press, Chicago, 1987.

[64] Thomas K Landauer, Peter W Foltz, and Darrell Laham, *An introduction to latent semantic analysis*, Discourse processes **25** (1998), no. 2-3, 259–284.

[65] Steve Lawrence, C. Lee Giles, and Ah Chung Tsoi, *What size neural network gives optimal generalization?: Convergence properties of backpropagation*, Tech. report, 1998.

[66] Quoc Le and Tomas Mikolov, *Distributed representations of sentences and documents*, International conference on machine learning, PMLR, 2014, pp. 1188–1196.

[67] Yann A. LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller, *Efficient backprop*, pp. 9–48, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[68] Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng, *Efficient sparse coding algorithms*, Advances in neural information processing systems, 2007, pp. 801–808.

[69] Y. Li, T. Cohn, and T. Baldwin, *Learning Robust Representations of Text*, arXiv preprint arXiv:1609.06082 (2016).

[70] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov, *Roberta: A robustly optimized bert pretraining approach*, (2019).

[71] George D. Magoulas, Michael N. Vrahatis, and George S. Androulakis, *Improving the convergence of the backpropagation algorithm using learning rate adaptation methods*, Neural Computation **11** (1999), no. 7, 1769–1796.

[72] Christopher D. Manning and Hinrich Schütze, *Foundations of statistical natural language processing*, MIT Press, 2001.

[73] Mehdi Hafezi Manshadi, James Allen, and Mary Swift, *Toward a universal underspecified semantic representation*, 13th Conference on Formal Grammar (FG 2008), Hamburg, Germany (2008).

[74] Scott Mcdonald and Michael Ramscar, *Testing the distributional hypothesis: The influence of context on judgements of semantic similarity*, In Proceedings of the 23rd Annual Conference of the Cognitive Science Society, 2001, pp. 611–6.

[75] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller, *Equation of state calculations by fast computing machines*, The journal of chemical physics **21** (1953), no. 6, 1087–1092.

[76] Risto Miikkulainen, *Natural language processing with subsymbolic neural networks*, Neural Network Perspectives on Cognition and Adaptive Robotics, Institute of Physics Publishing, 1997, pp. 120–139.

[77] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean, *Efficient estimation of word representations in vector space*, CoRR **abs/1301.3781** (2013).

[78] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean, *Distributed representations of words and phrases and their compositionality*, NIPS, Curran Associates, Inc., 2013, pp. 3111–3119.

[79] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig, *Linguistic regularities in continuous space word representations*, HLT-NAACL, 2013, pp. 746–751.

[80] Mridul K Mishra and Jaydeep Viradiya, *Survey of sentence embedding methods*, International Journal of Applied Science and Computations **6** (2019), no. 3, 592–592.

[81] Martin Fodslette Møller, *A scaled conjugate gradient algorithm for fast supervised learning*, Neural networks **6** (1993), no. 4, 525–533.

[82] Miguel Moreira and Emile Fiesler, *Neural networks with adaptive learning rate and momentum terms*, Tech. report, 1995.

[83] Marcel J. Nijman and Hilbert J. Kappen, *Efficient learning in sparsely connected boltzmann machines*, ICANN, 1996, pp. 41–46.

[84] E. Oja, *The nonlinear pca learning rule and signal separation: Mathematical analysis*, Helsinki University of Technology, 1995.

[85] Erkki Oja, *Principal components, minor components, and linear neural networks*, Neural networks **5** (1992), no. 6, 927–935.

[86] Bruno A Olshausen and David J Field, *Sparse coding with an overcomplete basis set: A strategy employed by v1?*, Vision research **37** (1997), no. 23, 3311–3325.

[87] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu, *Wavenet: A generative model for raw audio*, arXiv preprint arXiv:1609.03499 (2016).

[88] Jeffrey Pennington, Richard Socher, and Christopher D Manning, *Glove: Global vectors for word representation*, Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP), 2014, pp. 1532–1543.

[89] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer, *Deep contextualized word representations*, arXiv preprint arXiv:1802.05365 (2018).

[90] Nghia The Pham, Germán Kruszewski, Angeliki Lazaridou, Marco Baroni, et al., *Jointly optimizing word representations for lexical and sentential tasks with the c-phrase model*, Zong C, Strube M, editors. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers); 2015 Jul 26-31; Beijing, China. Stroudsburg (PA): Association for Computational Linguistics; 2015. p. 971-81, ACL (Association for Computational Linguistics), 2015.

[91] Jordan B. Pollack, *Recursive distributed representations*, Artificial Intelligence **46** (1990), 77–105.

[92] Ben Poole, Jascha Sohl-Dickstein, and Surya Ganguli, *Analyzing noise in autoencoders and deep networks*, arXiv preprint arXiv:1406.1831 (2014).

[93] John Porrill and James V Stone, *Undercomplete independent component analysis for signal separation and dimension reduction*, Tech. report, Citeseer, 1998.

[94] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever, *Improving language understanding by generative pre-training*, (2018).

[95] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al., *Language models are unsupervised multitask learners*, OpenAI blog **1** (2020), no. 8, 9.

[96] Juan Ramos et al., *Using tf-idf to determine word relevance in document queries*, Proceedings of the first instructional conference on machine learning, vol. 242, Citeseer, 2003, pp. 29–48.

[97] John W Ratcliff and David E Metzener, *Pattern-matching-the gestalt approach*, Dr Dobbs Journal **13** (1988), no. 7, 46.

[98] Suman V. Ravuri and Andreas Stolcke, *A comparative study of recurrent neural network models for lexical domain classification*, ICASSP, IEEE, 2016, pp. 6075–6079.

[99] Martin Rehn and Friedrich T Sommer, *A network that uses few active neurones to code visual input predicts the diverse shapes of cortical receptive fields.*, J Comput Neurosci **22** (2007), no. 2, 135–46.

[100] Nils Reimers and Iryna Gurevych, *Sentence-bert: Sentence embeddings using siamese bert-networks*, arXiv preprint arXiv:1908.10084 (2019).

[101] Martin Riedmiller and Heinrich Braun, *A direct adaptive method for faster backpropagation learning: The rprop algorithm*, Neural Networks, 1993., IEEE International Conference on, 1993, pp. 586–591.

[102] Herbert Robbins and Sutton Monro, *A stochastic approximation method*, The annals of mathematical statistics (1951), 400–407.

[103] AJ Robinson and Frank Fallside, *The utility driven dynamic error propagation network*, University of Cambridge Department of Engineering Cambridge (1987).

[104] Edmund T Rolls and Alessandro Treves, *The relative advantages of sparse versus distributed encoding for associative neuronal networks in the brain*, Network: computation in neural systems **1** (1990), no. 4, 407–421.

[105] E. Rosch, *Cognitive representations of semantic categories.*, Journal of Experimental Psychology: General **104** (1975), 192–223.

[106] Eleanor Rosch, *Principles of categorization*, Cognition and Categorization (Eleanor Rosch and B. B. Lloyd, eds.), Erlbaum, Hillsdale, NJ, 1978, pp. 27–48.

[107] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning internal representations by error propagation*.

[108] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams, *Learning representations by back-propagating errors*, nature **323** (1986), no. 6088, 533.

[109] M. Sahlgren, *The distributional hypothesis. from context to meaning: Distributional models of the lexicon in linguistics and cognitive science*, Rivista di Linguistica. Special issue of the Italian Journal of Linguistics **20** (2008).

[110] Ruslan Salakhutdinov and Geoffrey Hinton, *Semantic hashing*, Int. J. Approx. Reasoning **50** (2009), no. 7, 969–978.

[111] Ruslan Salakhutdinov and Geoffrey E. Hinton, *Deep boltzmann machines.*, Journal of Machine Learning Research - Proceedings Track **5** (2009), 448–455.

[112] Rico Sennrich, Barry Haddow, and Alexandra Birch, *Neural machine translation of rare words with subword units*, (2016).

[113] Richard Socher, Cliff Chiung-yu Lin, Andrew Y. Ng, and Christopher D. Manning, *Parsing natural scenes and natural language with recursive neural networks*, (2011).

[114] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts, *Recursive deep models for semantic compositionality over a sentiment treebank*, Proceedings of the 2013 conference on empirical methods in natural language processing, 2013, pp. 1631–1642.

[115] Mark Steedman, *The syntactic process*, MIT Press, Cambridge, MA, USA, 2000.

[116] Ilya Sutskever and Geoffrey Hinton, *Learning multilevel distributed representations for high-dimensional sequences*, Artificial Intelligence and Statistics, 2007, pp. 548–555.

[117] Ilya Sutskever, James Martens, and Geoffrey E. Hinton, *Generating text with recurrent neural networks*, ICML, 2011, pp. 1017–1024.

[118] Richard S. Sutton, *Two problems with backpropagation and other steepest-descent learning procedures for networks*, Proc. 8th annual conf. cognitive science society, 1986, pp. 823–831.

[119] Leonard Talmy, *Cognitive semantics: An overview*, Semantics: An International Handbook of Natural Language Meaning (Claudia Maienborn, Klaus von Heusinger, and Paul Portner, eds.), Walter de Gruyter, 2011, pp. 622–642.

[120] Talmy L., *Toward a cognitive semantics*, MIT Press, Cambridge, MA, 2000.

[121] Patrick V Taylor, J. Nicholas Hobbs, Javier Burroni, and Hava T. Siegelmann, *The global landscape of cognition: hierarchical aggregation as an organizational principle of human cortical networks and functions.*, Scientific reports **5** (2015), 18112.

[122] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin, *Attention is all you need*, Advances in neural information processing systems **30** (2017).

[123] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol, *Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion*, Journal of machine learning research **11** (2010), no. Dec, 3371–3408.

[124] Hanna M. Wallach, *Topic modeling: Beyond bag-of-words*, Proceedings of the 23rd International Conference on Machine Learning (New York, NY, USA), ICML '06, ACM, 2006, pp. 977–984.

[125] Dominic Widdows, Scott Cederberg, and Beate Dorow, *Visualisation techniques for analysing meaning*, Fifth International Conference on Text, Speech and Dialogue, Springer, 2002, pp. 107–115.

[126] Bogdan M. Wilamowski, *Neural network architectures and learning*, Industrial Technology, 2003 IEEE International Conference on, vol. 1, 2003, pp. –1.

[127] Ronald J Williams and David Zipser, *A learning algorithm for continually running fully recurrent neural networks*, Neural computation **1** (1989), no. 2, 270–280.

[128] Lei Wu, Steven CH Hoi, and Nenghai Yu, *Semantics-preserving bag-of-words models and applications*, IEEE Transactions on Image Processing **19** (2010), no. 7, 1908–1920.

[129] Shijie Wu, Ryan Cotterell, and Timothy J O'Donnell, *Morphological irregularity correlates with frequency*, Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pages 5117-5126. Florence, Italy, July 28 - August 2, 2019. Association for Computational Linguistics (2019).

[130] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le, *Xlnet: Generalized autoregressive pretraining for language understanding*, Advances in neural information processing systems **32** (2019).

[131] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al., *Big bird: Transformers for longer sequences*, Advances in Neural Information Processing Systems **33** (2020), 17283–17297.

[132] Z. Zainuddin, N. Mahat, and Y. Abu Hassan, *Improving the convergence of the backpropagation algorithm using local adaptive techniques.*, International Conference on Computational Intelligence, 2004, pp. 173–176.

[133] Matthew D. Zeiler, *Adadelta: an adaptive learning rate method*, arXiv preprint arXiv:1212.5701 (2012).

[134] Xiang Zhang and Yann LeCun, *Text understanding from scratch*, arXiv preprint arXiv:1502.01710 (2015).