Universidad Nacional Mayor de San Marcos
Universidad del Perú. Decana de América
Facultad de Ciencias Matemáticas
Escuela Profesional de Computación Científica

Modelo análisis de sentimientos: Word2Vec y un modelo BiRNR-LSTM para un discurso de odio

TESIS

Para optar el Título Profesional de Licenciado en Computación Científica

AUTOR

Josue Angel MAURICIO SALAZAR

ASESOR

Mg. Jhelly Reynaluz PÉREZ NÚÑEZ

Lima, Perú

2021



Reconocimiento - No Comercial - Compartir Igual - Sin restricciones adicionales

https://creativecommons.org/licenses/by-nc-sa/4.0/

Usted puede distribuir, remezclar, retocar, y crear a partir del documento original de modo no comercial, siempre y cuando se dé crédito al autor del documento y se licencien las nuevas creaciones bajo las mismas condiciones. No se permite aplicar términos legales o medidas tecnológicas que restrinjan legalmente a otros a hacer cualquier cosa que permita esta licencia.

Referencia bibliográfica

Mauricio, J. (2021). *Modelo análisis de sentimientos: Word2Vec y un modelo BiRNR-LSTM para un discurso de odio*. [Tesis de pregrado, Universidad Nacional Mayor de San Marcos, Facultad de Ciencias Matemáticas, Escuela Profesional de Computación Científica]. Repositorio institucional Cybertesis UNMSM.

Metadatos complementarios

Datos de autor	
Nombres y apellidos	Josue Angel Mauricio Salazar
Tipo de documento de identidad	DNI
Número de documento de identidad	75854603
URL de ORCID	https://orcid.org/0000-0002-8218-6923
Datos de asesor	
Nombres y apellidos	Jhelly Reynaluz Pérez Núñez
Tipo de documento de identidad	DNI
Número de documento de identidad	44076412
URL de ORCID	https://orcid.org/0000-0003-0717-8277
Datos del jurado	
Presidente del jurado	
Nombres y apellidos	Dra. María Natividad Zegarra Garay
Tipo de documento	DNI
Número de documento de identidad	09206994
Miembro del jurado 1	
Nombres y apellidos	Lic. Carlos Augusto Ruiz De La Cruz Melo
Tipo de documento	DNI
Número de documento de identidad	08249640
Miembro del jurado 2	
Nombres y apellidos	Mg. Jhelly Reynaluz Pérez Núñez
Tipo de documento	DNI
Número de documento de identidad	44076412
Datos de investigación	
Línea de investigación	No aplica

Grupo de investigación	No aplica
Agencia de financiamiento	Sin financiamiento
Ubicación geográfica de la investigación	Edificio: Universidad Nacional Mayor de San Marcos País: Perú Departamento: Lima Provincia: Lima Distrito: Cercado de Lima Latitud: -12.06041 Longitud: -77.08228
Año o rango de años en que se realizó la investigación	Julio 2021 - Setiembre 2021
URL de disciplinas OCDE	Informática y Ciencias de la Información https://purl.org/pe-repo/ocde/ford#1.02.00 Ciencias de la computación https://purl.org/pe-repo/ocde/ford#1.02.01

ACTA DE SUSTENTACIÓN DE TESIS EN LA MODALIDAD VIRTUAL PARA LA OBTENCIÓN DEL TÍTULO PROFESIONAL DE LICENCIADO(A) EN COMPUTACIÓN CIENTÍFICA (PROGRAMA DE TITULACIÓN PROFESIONAL 2021-I)

En la UNMSM – Ciudad Universitaria – Facultad de Ciencias Matemáticas, siendo las 11:20 horas del jueves 21 de octubre del 2021, se reunieron los docentes designados como Miembros del Jurado Evaluador (PROGRAMA DE TITULACIÓN PROFESIONAL 2021-I): Dra. María Natividad Zegarra Garay (PRESIDENTE), Mg. Carlos Augusto Ruiz De La Cruz Melo (MIEMBRO) y la Mg. Jhelly Reynaluz Pérez Núñez (MIEMBRO ASESOR), para la sustentación de la Tesis titulada: "**MODELO ANÁLISIS DE SENTIMIENTOS: WORD2VEC Y UN MODELO BiRNR-LSTM PARA UN DISCURSO DE ODIO**", presentado por el señor **Bachiller Josue Angel Mauricio Salazar**, para optar el Título Profesional de Licenciado en Computación Científica.

Luego de la exposición de la Tesis, la Presidente invitó al expositor a dar respuesta a las preguntas formuladas.

Realizada la evaluación correspondiente por los Miembros del Jurado Evaluador, el expositor mereció la aprobación **sobresaliente,** con un calificativo promedio de **diecisiete (17).**

A continuación, los Miembros del Jurado Evaluador dan manifiesto que el participante **Bachiller Josue Angel Mauricio Salazar** en vista de haber aprobado la sustentación de su Tesis, será propuesto para que se le otorgue el Título Profesional de Licenciado en Computación Científica.

Siendo las 12:00 horas se levantó la sesión firmando para constancia la presente Acta.

UNMSM
Firmado digitalmente por ZEGARRA GARAY Maria Natividad FAU 20148092282 soft
Motivo: Soy el autor del documento
Fecha: 03.11.2021 17:11:25 -05:00

Dra. María Natividad Zegarra Garay
PRESIDENTE

Mg. Carlos Augusto Ruiz De La Cruz Melo
MIEMBRO

Mg. Jhelly Reynaluz Pérez Núñez
MIEMBRO ASESOR

UNIVERSIDAD NACIONAL MAYOR DE SAN MARCOS

Universidad del Perú, DECANA DE AMÉRICA

FACULTAD DE CIENCIAS MATEMÁTICAS

Escuela Profesional de Computación Científica

"Año del Fortalecimiento de la Soberanía Nacional"

INFORME DE EVALUACIÓN DE ORIGINALIDAD

La directora de la Escuela Profesional de Computación Científica informa lo siguiente:

1. Operador del programa informático de similitudes: Dra. María Natividad Zegarra Garay.

2. Documento Evaluado:
   Tesis para obtener título de licenciado en Computación Científica, titulada «MODELO ANÁLISIS DE SENTIMIENTOS: WORD2VEC Y UN MODELO BIRNR-LSTM PARA UN DISCURSO DE ODIO».

3. Autor de la tesis: Josue Angel Mauricio Salazar.

4. Fecha de recepción de la tesis: 17 de octubre del 2021

5. Fecha de aplicación del programa informático de similitudes: 18 de octubre del 2021
   - Software utilizado: Turnitin

6. Configuración del programa detector de similitudes:
   - Excluye textos entrecomillados
   - Excluye bibliografía
   - Excluye cadenas menores a 40 palabras

7. Porcentaje de similitudes según programa detector de similitudes: dos por ciento (2%)

8. Fuentes originales de las similitudes encontradas:
   - Fuentes de internet: 2%
   - Publicaciones: 2%
   - Trabajos del estudiante: 1%

9. Calificación de originalidad:
   - Documento cumple criterios de originalidad, sin observaciones.

Lima, 6 de setiembre del 2022.

UNMSM
Firmado digitalmente por ZEGARRA GARAY Maria Natividad FAU 20148092282 soft
Motivo: Soy el autor del documento
Fecha: 07.09.2022 09:25:48 -05:00

Dra. María Natividad Zegarra Garay
Directora

/spv

Agradecimientos

Fiorella y Mateo, mi querida familia, por todo su amor, paciencia y comprensión. Ustedes son los verdaderos autores de este trabajo.

Resumen

El presente trabajo es el desarrollo de dos casos de estudio del Procesamiento de Lenguaje Natural usados en conjunto para poder generar un modelo de análisis de sentimientos, utilizando como fuente de datos *tweets* organizados en una tabla con 6000 datos etiquetados. Se buscará construir un modelo que pueda clasificar cuáles de estos mensajes contienen odio y cuáles no.

Primero, se hará un repaso de un modelo de representación vectorial para texto, llamado *word embeddings*, generados mediante el uso del algoritmo *Word2Vec*, se mostrará a detalle el mecanismo y resultados del mismo. Posteriormente, se construirá un modelo de clasificación binaria, el cual está compuesto por una arquitectura de redes neuronales recurrentes con unidades *Long Short-Term Memory* (LSTM) bidireccional, entrenado con los *embeddings* generados por el algoritmo *Word2Vec*, se explicará el uso de los parámetros utilizados en la configuración inicial del modelo.

Se mostrarán los resultados, haciendo uso de métricas de evaluación de nuestro modelo y por último, las respectivas conclusiones de nuestro estudio.

Palabras Clave: Procesamiento de Lenguaje Natural, word embeddings, Word2Vec, Análisis de sentimientos, Redes Neuronales Recurrentes.

Abstract

The present work is the development of two Natural Language Processing case studies used together in order to generate a sentiment analysis model, using as data source *tweets* organized in a table with 6000 labeled data. We will seek to build a model that can classify which of these messages contain hate and which do not.

First, we will review a vector representation model for text, called *word embeddings*, generated by using the *Word2Vec* algorithm, and we will show in detail its mechanism and results. Subsequently, a binary classification model will be built, which is composed of an architecture of recurrent neural networks with bidirectional *Long Short-Term Memory* (LSTM) units, trained with the *Word2Vec* generated by the *Word2Vec* algorithm, the use of the parameters used in the initial configuration of the model will be explained.

The results will be shown, making use of evaluation metrics of our model and finally, the respective conclusions of our study.

Keywords: Natural Language Processing, word embeddings, Word2Vec, Sentiment Analysis, Recurrent Neural Networks.

Índice general

1. Introducción **1**

2. Información del lugar donde se realizó la actividad **3**

3. Descripción de la actividad **4**

3.1. Presentación de la actividad 4

3.2. Trabajo relacionado 5

3.2.1. Estado del Arte del Uso de datos de Twitter para el análisis de sentimientos 6

3.2.2. Estado del Arte en la detección de discurso de odio en tweets . 6

3.3. Sustento teórico 8

3.3.1. Word Embeddings 8

3.3.1.1. One-Hot encodding 8

3.3.2. Word2Vec: Un método iterativo 9

3.3.2.1. Modelos de Lenguaje 10

3.3.2.2. El modelo Skip-Gram con Muestreo Negativo 11

3.3.3. Redes Neuronales Recurrentes (RNR) 16

3.3.3.1. Neurona Artificial 17
3.3.3.2. Backpropagation con Redes Neuronales Multicapa . . 18
3.3.3.3. Redes Neuronales Recurrentes(RNR) 21
3.3.4. RNR Bidireccional 24
3.3.5. LSTM: Long Short-Term Memory 25
3.3.6. Función de Pérdida Binary Cross-Entropy (BCE) 26
3.3.7. Optimizador ADAM 27
3.3.8. Métricas de Clasificación 28
3.3.8.1. Matriz de confusión 28
3.3.8.2. Principales Métricas 29
3.3.8.3. ROC 30
3.3.8.4. AUC 30
3.4. Importancia 31
3.5. Descripción Metodológica 31
3.5.1. Datos 31
3.5.2. Preprocesamiento de Datos 32
3.5.2.1. Limpieza de Datos 32
3.5.2.2. Tokenización 33
3.5.2.3. Stemming o Radicalización 34
3.5.2.4. Importancia de los *Hashtags* 34
3.5.3. Implementación de Word2Vec 35
3.5.4. Implementación del modelo RNR Bidireccional - LSTM(BiRNR-LSTM) 36
3.5.4.1. Arquitectura del modelo 36

3.6. Resultados 37

4. Conclusiones **42**

5. Recomendaciones **43**

Bibliografía **44**

Capítulo 1

Introducción

Muchas veces nos hemos hecho algunas preguntas, ¿Cómo funciona una red social?, ¿Cómo cierta información se vuelve una tendencia?, ¿Por qué cierta información genera controversia y/o división en un grupo de personas?. El análisis de sentimientos es un enfoque en el Procesamiento de Lenguaje Natural que se caracteriza por medir las emociones sobre un tema específico, es decir, identificar cual es la intención que hay detrás de un texto o representación visual.

Twitter, es una red social donde millones de personas interactúan utilizando ciertas herramientas de la aplicación para comunicar alguna idea, apreciación o comentario sobre un tema en particular. Dentro de este entorno, así como en otros medios sociales, se puede identificar una polarización respecto a que se piensa o se quiere decir sobre un comentario o *tweet*, que en ocasiones se vuelven tendencia. Es aquí donde nuestro estudio se ubicará y se definirá el análisis que buscará interpretar que es lo que pretende comunicar cierto comentario, réplica o reacción.

En el primer capítulo describiremos la actividad a realizar, definiendo los antecedentes y dando lugar, principalmente, a nuestro marco teórico. Se presentan los lineamientos del análisis de sentimientos, la interacción del Procesamiento de Lenguaje Natural en los medios sociales, en particular Twitter, definiremos los *word embeddings*, las diferentes representaciones vectoriales y el uso del algoritmo Word2Vec para la generación de estos. Se mostrará que hay detrás del algoritmo Word2Vec, los modelos Skip-Gram y Continuous Bag of Words(CBOW), y la arquitectura que sigue este modelo. Luego, se definirá el modelo de clasificación binaria, definiremos los preliminares de este modelo, la definición de Redes Neuronales Recurrentes(RNR), Long Short-Term Memory(LSTM), la arquitectura y diseño del modelo. Adicionalmente definiremos las métricas de evaluación para nuestro modelo, la función de pérdida Entropía Cruzada Binaria o por

sus siglas en inglés BCE y el optimizador Adam. Al final de este capítulo mostraremos el recorrido del modelo mediante el código potenciado por Gensim, Tensorflow y Keras. Estas 3 librerías serán las herramientas que necesitaremos para construir nuestro modelo. Se exponen los resultados de ambos modelos y las gráficas correspondientes.

En el capítulo 3 daremos paso a las conclusiones de nuestro trabajo y en el capítulo 4 las recomendaciones para futuros trabajos que comprenden la literatura de este análisis.

Capítulo 2

Información del lugar donde se realizó la actividad

Institución donde se desarrolló la actividad: Universidad Nacional Mayor de San Marcos

Periodo de duración de la actividad: 02/07/2021 - 15/10/2021

Razón social: Universidad Nacional Mayor de San Marcos

Dirección Postal: Cal. Germán Amezaga Nro. 375 Otros

Capítulo 3

Descripción de la actividad

3.1. Presentación de la actividad

En este capítulo trabajaremos en la construcción de nuestro marco teórico para proporcionar las herramientas detrás de nuestro Análisis de Sentimientos usando el algoritmo Word2Vec y un modelo de redes neuronales recurrentes con unidades LSTM. Se definen las siguientes secciones de este capítulo:

- Trabajo relacionado, se mencionarán los trabajos y revisiones relacionadas a la estructura de nuestro análisis.
- Sustento teórico, aquí definiremos la teoría detrás de nuestros modelos, el algoritmo Word2Vec, construcción de *word embeddings*, redes neuronales recurrentes, unidades Long Short-Term Memory(LSTM), la función de perdida Entropía Cruzada Binaria(BCE), el optimizador Adam, la arquitectura de nuestro modelo LSTM-Bidireccional, explicando a detalle las capas por las cuales pasarán nuestros datos para finalmente ser clasificados.
- Importancia, se dará paso a la justificación e importancia de la construcción de nuestro análisis.
- Descripción Metodológica, aquí colocaremos el paso a paso de la construcción de nuestro modelo haciendo uso de las herramientas servidas por el entorno de desarrollo de Gensim, Tensorflow y Keras, se colocarán las gráficas con las respectivas métricas de validación de nuestro modelo.
- Resultados, se colocarán los cuadros comparativos y resultados para cada modelo.

- Discusión, se discutirá sobre cuales son los parámetros óptimos dentro de nuestro modelo entrenado, la calidad y eficiencia del mismo.
- Recomendaciones, se brindará los lineamientos de mejora para posteriores trabajos dentro de la literatura.

3.2. Trabajo relacionado

Actualmente el análisis de sentimientos o *Sentiment Analysis*, por su traducción al inglés, es un campo estudiado en el ámbito académico así como también en el ámbito empresarial y político. Sin embargo, nosotros tenemos múltiples herramientas en nuestra vida cotidiana para poder expresar nuestras emociones o sentimientos respecto a alguna situación en particular. Es así como aparecen los medios sociales como la alternativa más útil de estos tiempos, causando un gran impacto en el consumo de información por las personas. En un estudio realizado por Monica Anderson (2021),por ejemplo, hoy en día en Estados Unidos se utiliza Youtube y Facebook en un 81 % y 69 %, respectivamente. Entre estos medios pondremos como ejemplo a Twitter, una red social donde millones de personas interactúan a diario, según el informe de Twitter del primer trimestre del año Twitter (2021a) a diario se genera un promedio de 500 millones de tweets, donde el el 63,7 % de los usuarios de Twitter son hombres, mientras que sólo el 36,3 % son mujeres. Según la edad de los usuarios, como ejemplo de este estudio, el 42 % de los usuarios estadounidenses de Twitter tienen entre 18 y 29 años, el 27 % de los usuarios tienen entre 30 y 49 años, cubriendo la mayoría de usuarios activos, inferimos así que un gran porcentaje de personas están utilizando esta red social en estos momentos.

El análisis de sentimientos es una serie de técnicas que tienen como objetivo graficar la intención o emoción respecto a un tema en específico, donde los elementos a estudiar son datos expuestos en forma de texto o alguna representación visual que exprese alguna idea, sentimiento o reacción respecto a la situación que se está atendiendo. Este enfoque es utilizado en el desarrollo de conexiones sociales, en marketing, servicio al cliente, etc. En la literatura encontramos trabajos cuyo desarrollo converge con el presente trabajo, Christos Troussas (2020) nos habla sobre conexiones en medios sociales, los avances en este campo mediante el uso de las técnicas del análisis de sentimientos, haciendo uso del Procesamiento de Lenguaje Natural y técnicas de Aprendizaje Máquina o *Machine Learning*, por su traducción al inglés.

Centrando nuestra visión en los modelos similares al análisis que expondremos, tenemos entonces que especificar que el medio social al que apuntaremos es Twitter, y para ello mostraremos un resumen breve de los trabajos relacionados en el análisis de sentimientos de datos ubicados en esta red social.

3.2.1. Estado del Arte del Uso de datos de Twitter para el análisis de sentimientos

Hemos mencionado que Twitter es una red social, cuyo objetivo es comunicar usuarios mediante sus herramientas, lo que envuelve a estas a interactuar, expresar sus emociones, informar, etc. Por ejemplo, en el ámbito de la información de la coyuntura política de una comunidad tenemos el trabajo de Chauhan et al. (2021) que resalta el estado del arte del análisis de sentimientos, el aprendizaje profundo y métodos de *word embeddings*, en el ámbito comercial o marketing tenemos a Kauffmann et al. (2019) quien logra utilizar la polaridad de los comentarios de clientes en Twitter para poder realizar recomendaciones basados en sus opiniones. Es posible también el uso de este enfoque para la gestión de prevención por desastres naturales, como se muestra en el artículo de Buscaldi et al. (2015). Una particularidad de estos trabajos es la capacidad definir un vocabulario o léxico para poder evidenciar los sentimientos de los datos que se explotan dentro de cada artículo.

3.2.2. Estado del Arte en la detección de discurso de odio en tweets

Nuestro trabajo va direccionado al análisis de un conjutos de datos, cuyo vocabulario o léxico tendrá concentrado la polaridad de los tweets que se recolectarán, es decir, que comentario es positivo o negativo. A esta definición se le llama en la literatura como 'Discurso de odio' o *Hate Speech* por su traducción al inglés. Básicamente consiste en detectar el odio en los comentarios vertidos dentro de la red social. Twitter (2021b) dentro de sus reglas de convivencia define el Discurso de Odio como una conducta de odio basada en ciertas características sociales como etnia, religión, origen, ideología de género,etc.

Existen trabajos que han estudiado este enfoque en la literatura, entre ellos tenemos a Jiang and Suzuki (2019), quien realiza un estudio con dos conjuntos de datos de una colección de tweets, ellos desarrollan una comparativa entre distintos modelos de extracción de características y redes neuronales recurrentes, donde obtienen una precisión de 62.08 % usando Word2Vec para la extracción de características y un 75.37 % de precisión usando el modelo BiRNN (Redes Neuronales Recurrentes Bidireccional).

En otro trabajo también podemos observar como los autores explotan el uso de los *woord embeddings* junto a un modelo basado en Redes Neuronales Recurrentes (RNR), Del Vigna et al. (2017) utiliza características morfo-sintácticas, análisis de sentimientos y un léxico de *word embeddings* para clasificar un conjunto de datos de 6000 comentarios de Facebook, usando SVM (Support Vector Machine) y LSTM (Long Short-Term Memory).

Artículo	Características	Modelo	Conjuto de Datos	Precisión
(Jiang and Suzuki, 2019)	TF-IDF vectors, Word2Vec, Doc2Vec and Bag of Words vectors (BoWV)	BIRNN-LSTM	9,925 datos etiquetados como racistas, sexista y no racista ni sexistas	75.37 %
(Del Vigna et al., 2017)	POS, word2vec embeddings, CBOW, n-gramas	SVM, LSTM	6502 comentarios de Facebook etiquetados	75.23 %
(Davidson et al., 2017)	n-gramas, TF-IDF, POS	Regresión lineal	24,802 tweets	91 %
(Alsafari et al., 2020)	Skip-gram, word2vec	CNN	19.5 millones de tweets	73.69 %

Cuadro 3.1: Cuadro comparativo de los distintos trabajos de la literatura en la detección de discurso de odio en tweets.

Dentro de la literatura también encontramos trabajos cuyos resultados superan algunas expectativas, tal es el caso de Davidson et al. (2017), quien logra obtener una precisión de 91 % utilizando un conjunto de datos etiquetados de 24,802 tweets, se usa TD-IDF(del inglés Term frequency – Inverse document frequency) para la extracción de características y un modelo de Regresión Lineal para su entrenamiento.

Mencionaremos también la capacidad de realizar un modelo basado en *word embeddings* y Redes Neuronales Convolucionales para realizar una clasificación múltiple de Discurso de Odio, Alsafari et al. (2020) nos presenta un estudio en el que utiliza un conjunto de datos de tweets en el lenguaje árabe, el cual etiqueta entre dos, tres y seis clases. Concluyendo en que los modelos que usan Skip-Gram para la representación vectorial del texto es más eficiente, y por el lado de las redes neuronales, elige el modelo CNN (del inglés Convolutional Neural Net) el cual tiene un mejor rendimiento alcanzando un 73.69 % de precisión.

En el cuadro 3.1, se muestra una comparación de los modelos descritos líneas arriba.

3.3. Sustento teórico

3.3.1. Word Embeddings

En el aprendizaje máquina y aprendizaje profundo, es necesario obtener un forma de representación numérica para los elementos a estudiar en el modelo seleccionado. Los *word embeddings* o vectores de palabras, del inglés *Word Vectors*, se utilizan a menudo como un componente fundamental para las tareas posteriores del Procesamiento de Lenguaje Natural, por ejemplo, la respuesta a preguntas, la generación de textos, la traducción, etc., por lo que es importante construir los lineamientos en cuanto a sus puntos fuertes y débiles. Chaubard et al. (2019)

3.3.1.1. One-Hot encodding

La representación *One-Hot*, es la manera más simple de representar un vector de palabras, ya que es un vector de $\mathbb{R}^{|V|\times 1}$ donde los índices están compuestos por elementos con valor 0 y 1. En esta notación $|V|$ es la dimensión del vocabulario en el texto. Por ejemplo si quisiéramos obtener la representación *One-Hot* de la siguiente oración:

El auto de papá es nuevo

se escribiría así:

$$w^{El} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, w^{auto} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, w^{de} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, w^{papá} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, w^{es} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, w^{nuevo} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

Sin embargo esta no es una representación óptima de las palabras en un texto, ya que no podríamos encontrar una similaridad con respecto al contexto de las otras palabras. Por ejemplo:

$$(w^{auto})^T w^{nuevo} = (w^{papá})^T w^{es} = 0$$

Es por ello que Chaubard et al. (2019) definen esta representación como 'localista', porque contiene información sólo sobre un punto de datos, y no da pistas sobre otros puntos. Es por ello que tenemos la necesidad de encontrar una representación más distribuida que pueda contener información de los otros puntos en el texto.

3.3.2. Word2Vec: Un método iterativo

Presentaremos ahora el núcleo de nuestro trabajo, el cual consiste en definir un nuevo enfoque respecto a la representación vectorial de las palabras de nuestro *corpus*. Los *embeddings* necesitan ser vectores cortos y densos, es decir en vez de construir vectores esparsa o con entradas con cero en su mayoría, los valores serán números reales que pueden ser también negativos. Serán cortos pues serán representados con una dimensión de 50-1000 entradas. Esto, a la larga nos ayudará con los errores clásicos, como el *overfitting* (Bashir et al. (2020)) en los resultados de nuestro modelo. Pues, intuitivamente es más sencillo para nuestro clasificador, entrenar vectores con baja dimensión, por ejemplo 300, respecto a vectores de alta dimnesión, por ejemplo 50,000.

La idea será diseñar un modelo cuyos parámetros sean los vectores de palabras o *embeddings*. Así, entrenaremos nuestro modelo respecto a un objetivo. En cada iteración correremos el modelo, evaluaremos los errores y se seguirá una regla de actualización que tienen la noción de penalizar los parámetros que han causado el error. Esta es una idea conocida como *Propagación hacia atrás* o *Backpropagation*, Rumelhart et al. (1988).

Es así como presentamos el modelo Word2Vec (Mikolov et al. (2013a)), presentado como una solución para el manejo de la representación vectorial distribuida de las palabras de un *corpus*. Word2Vec es un kit de herramientas que incluye lo siguiente:

- Dos algoritmos: *Continuous Bag of Words*(CBOW) y *Skip-Gram*. El primero, apunta a predecir la palabra central a partir de su vecindad o palabras que lo rodean, es decir, su contexto. Mientras que Skip-Gram hace lo contrario, predice la distribución(probabilidad) de su contexto o vecindad, a partir de la palabra central.
- Dos métodos de entrenamiento: Muestreo negativo(*Negative Sampling*) y Soft-Max jerárquico (*Hierarchical SoftMax*). El primero define un objetivo a a partir de ejemplos de muestreo negativo. El segundo, define un objetivo usando una estructura eficiente para el cálculo de la distribución de todo el vocabulario.

Antes de definir nuestro algoritmo y método de entrenamiento seleccionado, vamos a mencionar algunos modelos de organización de las palabras del *corpus*.

3.3.2.1. Modelos de Lenguaje

Lo primero, es generar un modelo que asigne una probabilidad a una secuencia de *tokens*. Por ejemplo:

Nuestro amigo se cayó de la escalera

Si definimos un modelo eficiente para esta oración, logrará tener una probabilidad alta. Por otro lado si tenemos la siguiente oración,

Cerdos vuelan, corren son juguetes

tendrá una probabilidad muy baja, ya que esta oración no tiene sentido alguno. Podemos colocar esta probabilidad en una secuencia de n palabras:

$$P(w_1, w_2, ..., w_n)$$

Si se asume que la ocurrencia entre palabras son completamente independiente, se definiría así un modelo unigrama o *1-gram*:

$$P(w_1, w_2, ..., w_n) = \prod_{i=1}^{n} P(w_i)$$

Sin embargo, hacer esto sería negligente ya que se sabe que la siguiente palabra depende de la secuencia de palabras anterior. Así el ejemplo de la frase sin sentido, tendría una probabilidad muy alta. Así definimos el modelo bigrama:

$$P(w_1, w_2, ..., w_n) = \prod_{i=2}^{n} P(w_i \mid w_{i-1})$$

Nuevamente aquí damos cuenta que solo nos estamos concentrando en las palabras vecinas, más no en una secuencia de palabras, por ejemplo, de una oración completa y compuesta. Sin embargo, ya se va teniendo una idea de como una serie o secuencia de *tokens* contiene una probabilidad, y como nuestros modelos pueden aprender de estas probabilidades. Más adelante se entenderá que estas probabilidades serán los pesos de entrada de nuestro modelo.

3.3.2.2. El modelo Skip-Gram con Muestreo Negativo

En el trabajo presente vamos a seleccionar el modelo Skip-Gram entrenado con Muestreo Negativo, por ello definiremos el paso a paso de este modelo.

Según , la intuición de Skip-Gram es la siguiente:

- Tomar una palabra objetivo y un contexto de palabras vecinas como ejemplos positivos.
- Aleatoriamente tomar palabras fuera del contexto de palabras vecinas para tomarlos como ejemplos negativos.
- Usar la regresión logística para entrenar un clasificador que pueda distinguir estos 2 casos.
- Finalmente, usar estos pesos aprendidos como nuestros *embeddings*.

Empezaremos explicando la intuición del modelo, y luego veremos como entrenarlo. Supongamos que tenemos la siguiente oración:

$$\text{... Juan mira(} \underbrace{se}_{c_1} \ \underbrace{colgó}_{c_2} \ \underbrace{el}_{c_3} \ \underbrace{gato}_{w} \ \underbrace{en}_{c_4} \ \underbrace{la}_{c_5} \ \underbrace{cortina}_{c_6} \text{) hay que bajarlo ...}$$

Nuestro objetivo es entrenar un clasificador, tal que, dado el par (w, c_i), donde w es nuestra palabra objetivo y c_i nuestro contexto de palabras vecinas, pueda retornar la probabilidad de que c_i este dentro del contexto real de w. Por ejemplo, si tenemos el par $(gato, martillo)$ y $(gato, cortina)$, el primero nos retornará un valor negativo, y el otro positivo.

La probabilidad de que la palabra c_i esté dentro del contexto real para w se representa del siguiente modo:

$$P(+ \mid w, c_i) \tag{3.1}$$

Mientras que la probabilidad de que la palabra c_i no esté dentro del contexto real para w se representa del siguiente modo:

$$P(- \mid w, c_i) = 1 - P(+ \mid w, c_i) \tag{3.2}$$

¿Cómo calculamos la probabilidad P?

Todo se basa en la similaridad con el *embedding*, una palabra es más próxima a la palabra objetivo si su *embedding* es similar al *embedding* de la palabra objetivo. Para calcular la similaridad entre dos o más *embeddings* se utilizará el **producto interno** de ellos.

$$Similaridad(w, c_i) \approx c \cdot w \tag{3.3}$$

Debemos decir que el producto interno $c \cdot w$ no es una probabilidad sino un número real entre $-$ínf a ínf, ya que los *embeddings* de Word2Vec pueden ser valores negativos, entonces el producto puede también ser negativo. Entonces, para volver a este valor una probabilidad, convertiremos este producto interno en una función $\sigma(x)$ **logística** o **sigmoide**, elemento fundamental de la regresión logística:

$$\sigma(x) = \frac{1}{1 + \exp(-x)} \tag{3.4}$$

De la ecuación (3.1) y (3.4), expresaremos la probabilidad de que la palabra c_i pertenezca al contexto real de la palabra w como:

$$P(+ \mid w, c_i) = \sigma(c \cdot w) = \frac{1}{1 + \exp(-c \cdot w)} \tag{3.5}$$

La función sigmoide retorna un número entre 0 y 1, pero si queremos que esto sea una probabilidad debe cumplir con que la suma de estos eventos(c_i pertenece al contexto real y c_i no pertenece al contexto real) sea 1. Por ello estimaremos que la probabilidad de que c_i pertenezca al contexto real de w es:

$$\begin{aligned} P(- \mid w, c_i) &= 1 - P(+ \mid w, c_i) \\ &= \sigma(-c \cdot w) = \frac{1}{1 + \exp(c \cdot w)} \end{aligned} \tag{3.6}$$

La ecuación (3.5) nos da la probabilidad para una sola palabra, sin embargo, existen muchas palabras dentro del contexto de la palabra objetivo, a esta longitud de palabras alrededor del contexto lo llamaremos *window*. Skip-Gram asumirá que todas las palabras dentro del contexto son independientes, esto nos permitirá multiplicar las probabilidades, dando paso a un modelo de lenguaje tal como se definió en la sección 3.3.2.1.

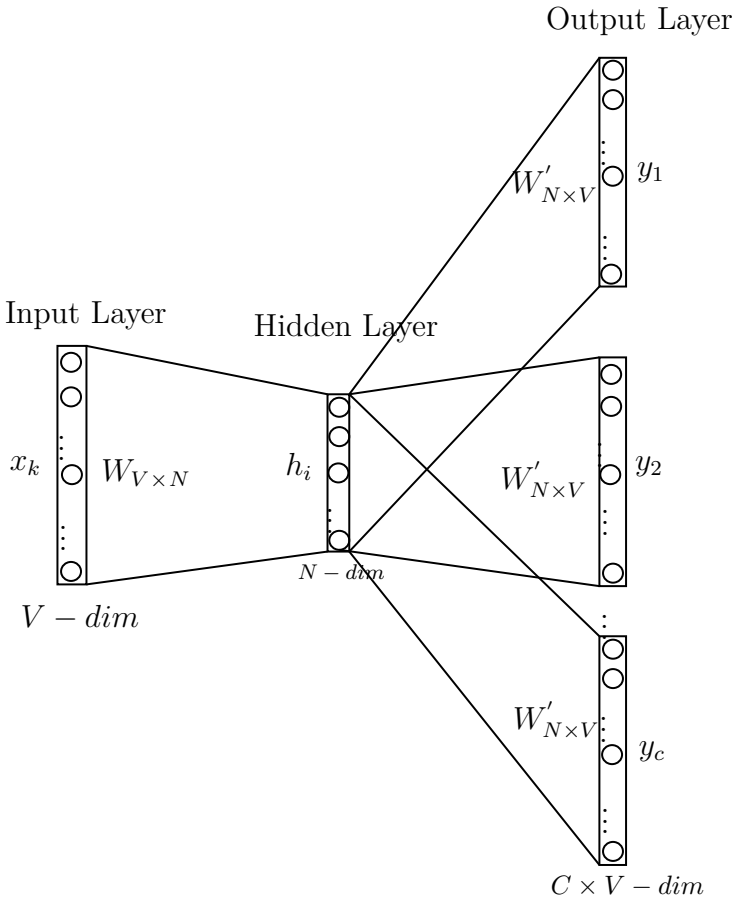


Figura 3.1: Este es un ejemplo que se acerca a la idea del aprendizaje del modelo Skip-Gram: Una manera de aprender las matrices de transferencia W y W'

$$P(+ \mid w, c_{1:L}) = \prod_{i=1}^{L} \sigma(-c_i \cdot w) \tag{3.7}$$

$$\log P(+ \mid w, c_{1:L}) = \sum_{i=1}^{L} \log \sigma(-c_i \cdot w) \tag{3.8}$$

En resumen, Skip-Gram entrena un clasificador probabilístico que, dada una palabra objetivo w y su contexto de *window* L de palabras $c_{1:L}$, asignará una probabilidad basada en cuan similar es este contexto a su palabra objetivo. La probabilidad está basada en aplicar la función logística al producto interno de los *embeddings* de la palabra objetivo con los de las palabras de su contexto. Para calcular esta probabilidad solo es necesario los *embeddings* para cada palabra objetivo y el contexto de palabras en el vocabulario $|V|$.

Aprendiendo los *embeddings* de Skip-Gram

De la figura 3.1 y de Mikolov et al. (2013a), se tienen los siguientes pasos para el aprendizaje de Skip-Gram:

1. En la primera capa ubicamos nuestra palabra central $x_k \in \mathbb{R}^{1\times|V|}$ en forma de un vector *one-hot*, donde V es el vocabulario de nuestro conjunto de datos.
2. Se introduce la matriz de pesos de entrada $W_{|V|\times N}$, donde N es la dimensión de las características para cada palabra.
3. Se realiza el producto interno entre x_k y $W_{|V|\times N}$, ya que x_k es un vector *one-hot*, se puede decir que la i-ésima fila de $W_{|V|\times N}$ será la salida de nuestra capa oculta, conformada por un vector $H \in \mathbb{R}^{1\times N}$.
4. En la capa de salida aplicaremos el producto interno ente $H \in \mathbb{R}^{1\times N}$ y $W'_{N\times|V|}$, obteniendo el vector $U \in \mathbb{R}^{1\times|V|}$.
5. Lo siguiente será encontrar la probabilidad de cada vector, para ello utilizaremos la función *SoftMax*. En cada iteración vamos a tener un vector U de tipo *one-hot*.
6. La palabra con la probabilidad más alta es el resultado a escoger, y si esta palabra no se ubica en la posición del contexto dado, entonces utilizaremos la propagación hacia atrás(o del inglés *backpropagation*) para modificar los pesos de nuestras matrices W y W'.

Estos pasos serán ejecutados para cada palabra x_k presente en el vocabulario y cada palabra será pasado k veces. Así, podemos ver que la propagación hacia adelante(o del inglés *forward propagation*) será procesado $|V| \times k$ veces en cada ciclo, o de la literatura de las RNR, en cada *epoch*.

Ahora bien, de este procedimiento debemos decir que, si bien es cierto, se reduce la dimensión del procesamiento de cada ciclo, ya que si N es 300(número de características) es más eficiente que $N = |V|$. Pero, en ocasiones es complicado darle un valor a N y al *window* del contexto C. Además la función *SoftMax* es computacionalmente costosa, pues es conocido que la función exponencial es computacionalmente costosa, Kozubek et al. (2016). Esto ocasiona que el tiempo requerido para entrenar este modelo sea alto.

Por ello, es necesario introducir una idea que se aproxime a este procedimiento buscando mayor eficiencia en los cómputos del modelo.

Negative Sampling

Para cada paso del entrenamiento, en vez de iterar sobre el vocabulario entero $|V|$, nosotros solo tomaremos una muestra de ejemplos negativos. Para ello realizaremos un

muestreo(o del inglés *sample*) desde una dsitrbución ruidosa $P_n(w)$ cuyas probabilidades coinciden con el ordenamiento de la frecuencia del vocabulario.

Negative Sampling, Mikolov et al. (2013b). Este enfoque se concentra en usar más ejemplos negativos que positivos. Así que para cada par (w, c_{pos}) del conjunto de entrenamiento, crearemos k ejemplos negativos, cada uno consiste de la suma de la palabra objetivo w y una palabra *ruidosa* c_{neg}. Una palabra *ruidosa* es una palabra del léxico, tal que está no sea la palabra objetivo w.

Dado el conjunto de ejemplos positivos y negativos, y un conjunto inicial de *embeddings*, el propósito del algoritmo es ajustar estos *embeddings* a los siguientes principios:

- Maximizar la similitud de la palabra objetivo w, los pares de palabras contexto (w, c_{pos}) de los ejemplos positivos.
- Minimizar la similitud de los pares (w, c_{neg}) de los ejemplos negativos.

Si consideramos un par (w, c_{pos}) con sus k palabras *ruidosas* $w_{neg_1}, w_{neg_2}, ..., w_{neg_k}$, podemos considerar estos dos objetivos como la siguiente **función de perdida** L a ser minimizada(considerar el $-$).

$$\begin{aligned} L_{CE} &= -\log\left[P(+ \mid w, c_{pos}) \prod_{i=1}^{k} P(- \mid w, c_{neg_i})\right] \\ &= -\left[\log P(+ \mid w, c_{pos}) + \sum_{i=1}^{k} \log P(- \mid w, c_{neg_i})\right] \\ &= -\left[\log P(+ \mid w, c_{pos}) + \sum_{i=1}^{k} \log(1 - P(+ \mid w, c_{neg_i}))\right] \\ &= -\left[\log \sigma(c_{pos}\dot{w}) + \sum_{i=1}^{k} \log \sigma(-c_{neg_i}\dot{w})\right] \end{aligned} \tag{3.9}$$

De aquí el primer término expresa que la palabra c_{pos} del contexto real, tenga una alta probabilidad de ser parte del *window*, y el segundo término expresa que buscamos asignar a cada una de estas palabras *ruidosas* una alta probabilidad de no ser parte del *window* de nuestra palabra objetivo, todas multiplicadas ya que se asume son independientes una de la otra.

Por tanto de los principios anteriores, se busca maximizar el producto interno de la palabra con las palabras del contexto real o positivo, y minimizar el producto interno de la palabra con las k palabras negativas que no son parte del contexto real. Minimizaremos la función de perdida L utilizando la gradiente descendiente estocástica.

Realizaremos entonces la gradiente, tomando las derivadas respecto de los diferentes *embeddings*:

$$\frac{\partial L_{CE}}{\partial c_{pos}} = \left[\sigma(c_{pos} \cdot w) - 1\right] w \tag{3.10a}$$

$$\frac{\partial L_{CE}}{\partial c_{neg}} = \left[\sigma(c_{neg} \cdot w)\right] w \tag{3.10b}$$

$$\frac{\partial L_{CE}}{\partial w} = \left[\sigma(c_{pos} \cdot w) - 1\right] c_{pos} + \sum_{i=1}^{k} \left[\sigma(c_{neg_i} \cdot w)\right] c_{neg_i} \tag{3.10c}$$

Las ecuaciones de actualización que van del paso t a $t+1$ en la gradiente descendiente estocástica son:

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta \left[\sigma(c_{pos}^{t} \cdot w) - 1\right] w \tag{3.11a}$$

$$c_{pos}^{t+1} = c_{pos}^{t} - \eta \left[\sigma(c_{neg}^{t} \cdot w)\right] w \tag{3.11b}$$

$$w^{t+1} = w^{t} - \eta \left[\sigma(c_{pos} \cdot w^{t}) - 1\right] c_{pos} + \sum_{i=1}^{k} \left[\sigma(c_{neg_i} \cdot w^{t})\right] c_{neg_i} \tag{3.11c}$$

De esta manera nuestra función de perdida se convierte en un problema de regresión, que comenzó con dos matrices W y C inicializadas aleatoriamente, y luego mediante el uso de la gradiente descendiente actualizar W y C de forma que se maximice el objetivo.

De aquí, se tiene en breve que nuestro modelo Skip-Gram tiene como propósito entrenar dos *embeddings* para cada palabra x_k del vocabulario V: el ***embedding* objetivo** w_i y el ***embedding* contexto** c_i, almacenados en la **matriz objetivo** W y **matriz contexto** C, respectivamente.

3.3.3. Redes Neuronales Recurrentes (RNR)

Las redes neuronales recurrentes, también conocidas como RNNs, del inglés Recurrent Neural Networks, son una clase de redes neuronales que permite a las salidas (*outputs*)

previas sean usadas como entradas (*inputs*) mientras tengan estados ocultos. Previo a ello, vamos a definir términos básicos de redes neuronales.

3.3.3.1. Neurona Artificial

Sean $\{x_1, x_2, ..., x_k\}$ son los valores de entrada (*inputs*), $\{w_1, w_2, ..., w_k\}$ son pesos; y es un valor de salida escalar; y f es la función de activación.

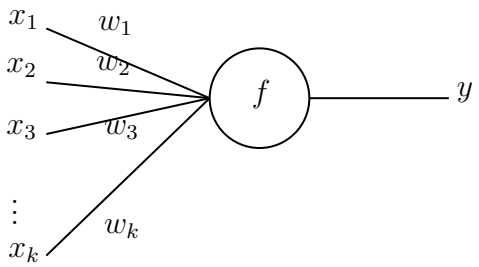


Figura 3.2: Una neurona artificial

La unidad trabaja de la siguiente manera:

$$y = f(u) \tag{3.12}$$

Donde u es un número escalar, al cual llamaremos *entrada* o *input* de la neurona, u se define como:

$$u = \sum_{i=0}^{K} w_i x_i \tag{3.13}$$

Seleccionaremos la **función logística** o función sigmoide, como nuestra función de activación $f(u)$.

$$\sigma(u) = \frac{1}{1 + \exp -u} \tag{3.14}$$

La función logística es una función suave y diferenciable y su valor de salidad y siempre estará entre 0 y 1. Se utilizará las gradiente descendiente estocástica como algoritmo de aprendizaje de este modelo. Definimos la función de error convenientemente:

$$E = \frac{1}{2}(t - y)^2 \tag{3.15}$$

Tomando las derivadas respecto a w_i:

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial w_i} \\ &= (y - t) \cdot y(1 - y) \cdot x_i \end{aligned} \tag{3.16}$$

Luego, aplicamos nuestra gradiente descendiente estocástica, obteniendo así:

$$\mathbf{w}^{new} = \mathbf{w}^{old} - \eta \cdot (y - t) \cdot y(1 - y) \cdot \mathbf{x} \tag{3.17}$$

3.3.3.2. Backpropagation con Redes Neuronales Multicapa

Sea una red neuronal multicapa con una capa de entrada $\{x_k\} = \{x_1, ..., x_k\}$, una capa oculta $\{h_i\} = \{h_1, ..., h_N\}$ y una capa de salida $\{y_j\} = \{y_1, ..., y_M\}$, utilizaremos los subíndices k, i, j para las unidades de la capa de entrada, oculta y de salida, respectivamente.

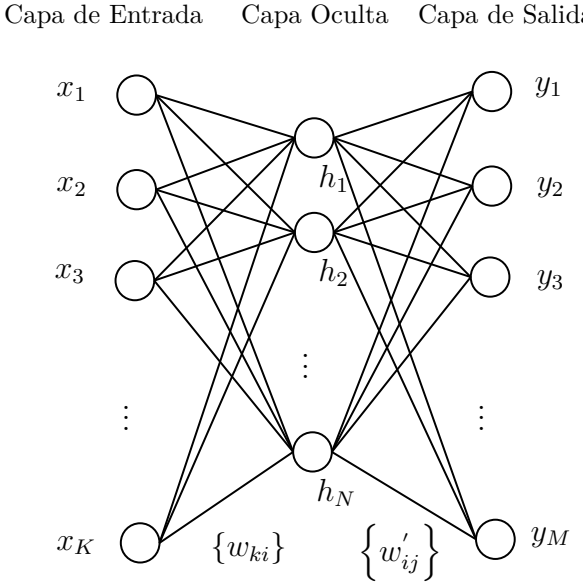


Figura 3.3: Una red neuronal multicapa con una sola capa oculta

Denotaremos u_i y $u_j^{'}$ para denotar las unidades de entrada de la capa oculta y capa de salida, respectivamente. Nosotros buscamos derivar la ecuación de actualización para aprender los pesos w_{ki} entre las capa de entrada y oculta, y $w_{ij}^{'}$ entre la capa oculta y de salida.

Suponiendo que la unidades calculadas (unidades de la capa oculta y capa de salida) usan la función logística $\sigma(u)$ como la función de activación. Por tanto, para una unidad h_i en la capa oculta, su salida está definida como:

$$h_i = \sigma(u_i) = \sigma\left(\sum_{k=1}^{K} w_{ki} x_k\right) \tag{3.18}$$

De igual manera, para una unidad y_j en la capa de salida, su salida está definida como:

$$y_j = \sigma(u_j^{'}) = \sigma\left(\sum_{i=1}^{N} w_j^{'} h_i\right) \tag{3.19}$$

Introduciremos la siguiente función de error de la suma cuadrática dada por:

$$E(\boldsymbol{x}, \boldsymbol{t}, \boldsymbol{W}, \boldsymbol{W'}) = \frac{1}{2}\sum_{j=1}^{M}(y_j - t_j)^2 \tag{3.20}$$

donde $\boldsymbol{W} = \{w_{ki}\}$, una matriz de pesos $K \times N$ (capa entrada - capa oculta), y $\boldsymbol{W'} = \{w_{ij}^{'}\}$, una matriz de pesos $N \times M$ (capa oculta - capa salida). $\boldsymbol{t} = \{t_1, ..., t_M\}$, un vector de dimensión M, etiquetas convenientemente definidas de la salida.

Para obtener las ecuaciones de actualización para w_{ki} y $w_{ij}^{'}$, nosotros simplemente necesitamos tomar las derivadas de la función de error E con respecto a cada peso, respectivamente. Para hacer la derivación más sencilla, empezaremos a calcular la derivada desde la capa más a la derecha, es decir, desde la capa de salida. Realizaremos este procedimiento con dirección a la izquierda. Para cada capa, calcularemos las derivadas del error con respecto a los valores de salida, entrada y pesos respectivamente.

Comenzaremos con la capa de salida, calculando la derivada del error con respecto a los valores de salida:

$$\frac{\partial E}{\partial y_j} = y_j - t_j \tag{3.21}$$

Lo siguiente será calcular las derivadas con respecto a las unidades de entrada de la capa de salida:

$$\frac{\partial E}{\partial u_j^{'}} = \frac{\partial E}{\partial y_j} \cdot \frac{\partial y_j}{\partial u_j^{'}} = (y_j - t_j) \cdot y_j(1 - y_j) := \mathrm{EI}_j^{'} \tag{3.22}$$

El tercer paso es calcular pas drivadas del error con respecto a los pesos entre la capa oculta y la capa de salida:

$$\frac{\partial E}{\partial w_{ij}^{'}} = \frac{\partial E}{\partial u_j^{'}} \frac{\partial u_j^{'}}{\partial w_{ij}^{'}} = \mathrm{EI}_j^{'} \cdot h_i \tag{3.23}$$

Así, obtenemos la ecuación de actualización para los pesos entre la capa oculta y la capa de salida.

$$\begin{aligned} w_{ij}^{'\,(\mathrm{new})} &= w_{ij}^{'\,(\mathrm{old})} - \eta \cdot \frac{\partial E}{\partial w_{ij}^{'}} \\ &= w_{ij}^{'\,(\mathrm{old})} - \eta \cdot \mathrm{EI}_j^{'} \cdot h_i \end{aligned} \tag{3.24}$$

donde $\eta > 0$ es el ratio de aprendizaje.

Repetiremos los tres pasos para obetener las ecuaciones de actualizacion para los pesos de la capa previa, de esto la noción del *backpropagation* o propagación hacia atrás.

El primer paso es calcular las derivadas del error con respecto a las unidades de salida de la capa oculta. Note que las salidas de la capa oculta están realcionadas con todas las unidades de la capa de salida.

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{M} \frac{\partial E}{\partial u_j^{'}} \frac{\partial u_j^{'}}{\partial h_i} = \sum_{j=1}^{M} \mathrm{EI}_j^{'} \cdot w_{ij}^{'} \tag{3.25}$$

El segundo paso, calcular las derivadas con respecto a las unidades de entrada de la capa oculta. Este valor será denotado por EI_i.

$$\frac{\partial E}{\partial u_i} = \frac{\partial E}{\partial h_i} \frac{\partial h_i}{\partial u_i} = \sum_{j=1}^{M} \mathrm{EI}_j^{'} \cdot w_{ij}^{'} \cdot h_i(1 - h_i) := \mathrm{EI}_i \tag{3.26}$$

Lo siguiente, encontrar las derivadas del error con respecto a los pesos entre la capa de entrada y la capa oculta.

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial u_i}\frac{\partial u_i}{\partial w_{ki}} = \mathrm{EI}_i \cdot x_k \tag{3.27}$$

Finalmente, obtenemos la ecuación de actualización para los pesos entre la capa de entrada y la capa oculta.

$$\begin{aligned} {w_{ki}}^{(\mathrm{new})} &= {w_{ki}}^{(\mathrm{old})} - \eta \cdot \frac{\partial E}{\partial w_{ki}} \\ &= {w'_{ki}}^{(\mathrm{old})} - \eta \cdot \mathrm{EI}_i \cdot x_k \end{aligned} \tag{3.28}$$

De este ejemplo, podemos ver que para los resultados intermedios $(\mathrm{EI})'_j$ cuando calculamos las derivadas para una capa pueden ser usadas por la capa previa. Si hubiesen más capas antes de llegar a la capa de entrada, entonces $(\mathrm{EI})_i$ sería reutilizado para poder calcular la cadenas de derivadas eficientemente.

De la ecuación 3.26, se puede decir que el término $\sum_{j=1}^{M} \mathrm{EI}'_j \cdot w'_{ij}$ es el error de la unidad h_i de la capa oculta. Interpretamos este error como el error *propagado hacia atrás* o *"back-propagated"* de la siguiente capa, y esta propagación podría ir más lejos si la tuviese muchas más capas ocultas.

3.3.3.3. Redes Neuronales Recurrentes(RNR)

Una red neuronal recurrente es una red que contiene un ciclo dentro de sus conexiones. Es decir, una red donde el valor de una de sus unidades es directa o indirectamente dependiente de sus propias salidas, en su valor de entrada. Aunque estas redes son avanzadas, son difíciles de razonar y entrenar. Sin embargo, dentro de la clase general de redes recurrentes hay arquitecturas limitadas que han demostrado ser extremadamente eficaces cuando se aplican al procesamiento de lenguaje natural, escrito o hablado. Jurafsky (2020)

Así como los modelos de traducción convencionales, donde solo un para un rango o *window* de palabras contexto debía ser considerado para condicionarl el modelo de lenguaje, las Redes Neuronales Recurrentes (RNR) son capaces de condicionar el modelo para todo el contexto dentro del *corpus*.

En la figura 3.4 se introduce la arquitectura de una RNR donde cada caja rectangular horizontal es una capa oculta en el paso de tiempo t. Para cada capa se configura

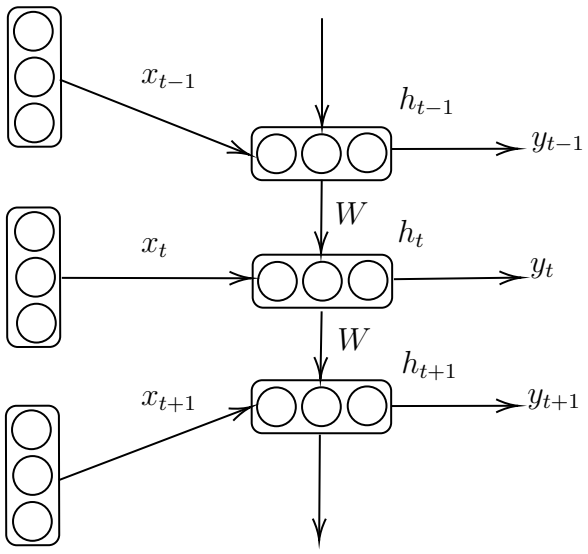


Figura 3.4: Una red neuronal recurrente (RNR), se muestran 3 pasos a través del tiempo.

un número de neuronas, cada una desarrolla una operación matricial lineal sobre sus entradas seguidas por una operación no-lineal, por ejemplo, $tan(h)$.

Para cada paso de tiempo, existen dos entradas o *inputs* para la capa oculta: la salida de la capa previa h_{t-1} en el paso de tiempo $t-1$ y la entrada para el paso de tiempo actual x_t. El primer *input* será multiplicado con la matriz de pesos $W^{(hh)}$ y el segundo, con la matriz de pesos $W^{(hx)}$ para producir las características de salida h_t que se multiplican por una matriz de pesos $W^{(S)}$ y se someten a un proceso *SoftMax* sobre el vocabulario para obtener la salida de predicción $\hat{y}$ de la siguiente palabra.

$$h_t = \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right) \tag{3.29}$$

$$\hat{y} = SoftMax(W^{(S)}h_t) \tag{3.30}$$

La figura 3.5 ilustra las entradas y salidas de una neurona singular.

Lo interesante de esto es que las mismas matrices de pesos $W^{(hh)}$ y $W^{(hx)}$ son aplicadas repetidamente en cada paso de tiempo. Así, el número de parámetros del modelo a aprender es menor, y más aún, es independiente de la longitud de secuencia de entrada, así deja sin efecto el problema de dimensionalidad.

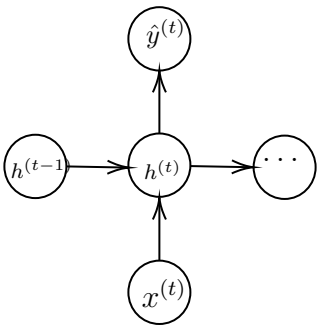


Figura 3.5: Las entradas y salidas a una neurona en una RNR.

Detallaremos cada parámetro de la red:

- $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$ los vectores de palabras corresponden a un corpus de T palabras.
- $h_t = \sigma\left(W^{(hh)}h_{t-1} + W^{(hx)}x_{[t]}\right)$: la regla para calcular las características de salida en la capa oculta en cada paso de tiempo t.
  - $x_t \in \mathbb{R}^d$: vector de palabras de entrada en el tiempo t.
  - $W^{(hx)} \in \mathbb{R}^{D_h \times d}$: matriz de pesos usado para condicionar la salida de vector de palabras, x_t.
  - $W^{(hh)} \in \mathbb{R}^{D_h \times D_h}$: matriz de pesos para condicionar la salida del paso previo, h_{t-1}.
  - $h_{t-1} \in \mathbb{R}^{D_h}$: salida de la función no-lineal del paso previo, $t-1$. $h_0 \in \mathbb{R}^{D_h}$ es un vector de inicialización para la capa oculta del paso $t = 0$.
  - $\sigma(\cdot)$: la función no-lineal(aquí, función sigmoide).
- $\hat{y} = SoftMax(W^{(S)}h_t)$: la distribución de probabilidad de la salida sobre el vocabulario en cada paso de tiempo t. Básicamente, $\hat{y}$ es la siguiente palabra predicha dado el resultado del contexto h_{t-1} y el último vector observado x_t. Aquí, $W^{(S)} \in \mathbb{R}^{|V| \times D_h}$ y $\hat{y} \in \mathbb{R}^{|V|}$ donde $|V|$ es la dimensión del vocabulario.

La función de perdida usada a menudo en las RNR es el error de cruce de entropía, visto anteriormente.

$$J^{(t)}(\theta) = -\sum_{j=1}^{|V|} y_{t,j} \times \log(\hat{y}_{t,j}) \tag{3.31}$$

El error de cruce de entropía sobre el *corpus* de tamaño T es:

$$J = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{j=1}^{|V|} y_{t,j} \times \log(\hat{y}_{t,j}) \tag{3.32}$$

3.3.4. RNR Bidireccional

Hemos visto aquellas RNR cuya condición es aprender de las palabras pasadas para predecir las siguientes palabras en una secuencia. Es posible realizar una predicción basada en palabras **futuras**, haciendo que el modelo RNR lea a través de el corpus en dirección opuesta. Irsoy (2013), presenta un modelo de red neuronal recurrente bidireccional, para cada paso de tiempo t esta red mantiene dos capas ocultas: una para la propagación de izquierda a derecha, y otra para la propagación de derecha a izquierda. El resultado de la clasificación final, $\hat{y}_t$, generado al combinar los resultados producidos por ambas capas ocultas de la RNR. Mostramos en la figura 3.6 un pequeño ejemplo de una RNR bidireccional.

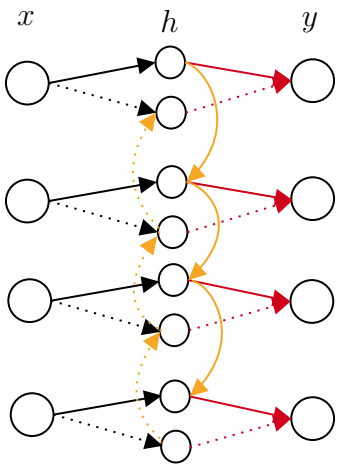


Figura 3.6: Un modelo de RNR Bidireccional

Las ecuaciones (3.33) y (3.34), muestran la formulación matemática detrás de la configuración de una capa oculta de la RNR Bidireccional. La única diferencia entre estas dos relaciones es en la dirección del recorrido a través del *corpus*. La (3.35) nos muestra la relación de clasificación usada para predecir la siguiente palabra mediante un resumen de las palabras pasadas y futuras.

$$\vec{h_t} = f\left(\vec{W}_{X_t} + \vec{V}\vec{h}_{t-1} + \vec{b}\right) \tag{3.33}$$

$$\overleftarrow{h_t} = f\left(\overleftarrow{W}_{X_t} + \overleftarrow{V}\,\overleftarrow{h}_{t-1} + \overleftarrow{b}\right) \tag{3.34}$$

$$\hat{y}_t = g\left(Uh_t + c\right) = g\left(U\left[\overrightarrow{h}_t; \overleftarrow{h}_t\right] + c\right) \tag{3.35}$$

3.3.5. LSTM: Long Short-Term Memory

Long Short-Term Memory o Redes de gran memoria de corto plazo (LSTM), son otro tipo de unidades de activación compleja. La motivación de usar LSTM es que, si bien es cierto teóricamente las RNR pueden capturar dependencias a largo plazo, en la práctica es difícil poder lograr ello en la fase de entrenamiento.

Vamos a tomar una primera vista de la formulación matemática de las unidades LSTM antes de detallar la intuición detrás de este diseño.

$$\begin{array}{lcll} i_t & = & \sigma(W^{(i)}x_t + U^{(t)}h_{t-1}) & \text{(Input gate)} \\ f_t & = & \sigma(W^{(f)}x_t + U^{(t)}h_{t-1}) & \text{(Forget gate)} \\ O_t & = & \sigma(W^{(O)}x_t + U^{(O)}h_{t-1}) & \text{(Output/Exposure gate)} \\ \tilde{c}_t & = & tanh(W^{(c)}x_t + U^{(c)}h_{t-1}) & \text{(New memory cell)} \\ c_t & = & f_t \circ c_{t-1} + i_t \circ \tilde{c}_t & \text{(Final memory cell)} \\ h_t & = & O_t \circ tanh(c_t) & \end{array}$$

De la figura 3.7, describiremos las etapas de la arquitectura de una unidad LSTM:

1. **Generación de memoria nueva o New memory generation**: Se usa la palabra *input* x_t y el estado oculto anterior h_{t-1} para generar una nueva memoria $\tilde{c}_t$ la cual incluye los aspectos de la nueva palabra x^t.

2. **Celda de Entrada o Input Gate**: La Celda de entrada utiliza la palabra de entrada y el estado oculto anterior para determinar si vale la pena conservar la entrada y, por lo tanto, se utiliza para abrir la nueva memoria. Por tanto, produce i_t como indicador de esta información.

3. **Celda de Olvido o Forget Gate**: Esta celda es similar a la celda de entrada, a excepción que esta no realiza una determinación de la utilidad de la palabra de entrada, sino hace una evaluación sobre si la celda de memoria pasa es útil para el cálculo de la celda de memoria actual. Así, la celda de olvido mira la palabra de entrada y el estado pasado oculto para producir f_t.

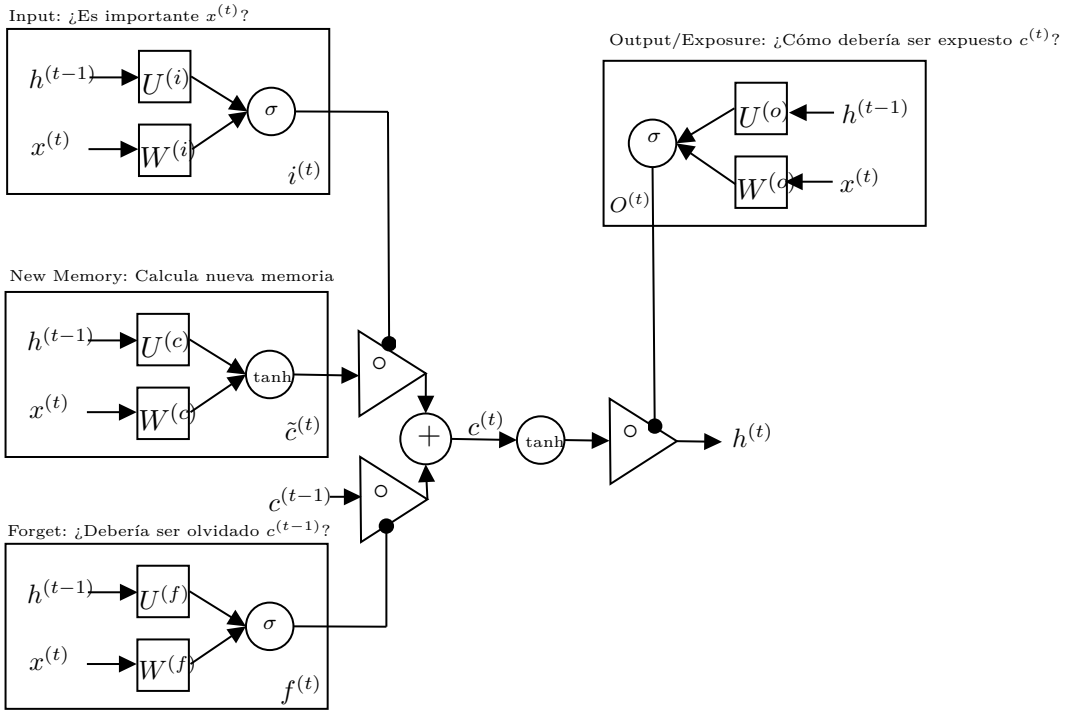


Figura 3.7: Arquitectura detallada de una unidad LSTM.

4. **Generación de la memoria final o Final Memory Generation**: Esta etapa primero toma el resultado de la celda de olvido f_t y de acuerdo a ello, olvida la memoria pasada c_{t-1}. Similarmente, este toma el resultado de la celda de entrada i_t y de acuerdo a ello genera la nueva memoria $\tilde{c}_t$. Se suman ambos resultados para obtener la memoria final c_t.

5. **Celda de Exposición/Salida o Output/Exposure Gate**: El objetivo de esta celda es separar la memoria final del estado oculto. La memoria final c_t contiene mucha información que no necesariamente se requiere almacenar en el estado oculto. Los estados ocultos son usados en cada celda de una unidad LSTM y por tanto, esta celda hace la evaluación respecto a que partes de la memoria c_t necesitan ser expuestas en el estado oculto h_t. La señal producida para indicar esto es o_t, usada para la decisión junto a la tanh de la memoria final.

3.3.6. Función de Pérdida Binary Cross-Entropy (BCE)

La función de pérdida de Cruce de Entropía Binaria o Binary Cross-Entropy (BCE), es una función usada en tareas de clasificación binaria. Estos son tareas que responden a una pregunta con solo 2 respuestas: Si o No, A o B, 0 o 1, etc. Formalmente, esta función de pérdida es igual a la media de la pérdida del cruce de entropía categórico en

muchas tareas de 2 categorías.

$$CE = -\sum_{i=1}^{C'=2} t_i \log(f(s_i)) = -t_1 \log(f(s_1)) - (1-t_1) log(1-f(s_1)) \tag{3.36}$$

Donde $f(s_i) = \frac{1}{1+\exp(-s_i)}$, la función sigmoide.

La función de pérdida, también puede ser expresada como:

$$CE = \begin{cases} -\log(f(s_1)) & \text{si } t_1 = 1 \\ -\log(1-f(s_1)) & \text{si } t_1 = 0 \end{cases} \tag{3.37}$$

Donde $t_1 = 1$ significa que la clase $C_1 = C_i$ es positivo para esta muestra.

En este caso, la función de activación no depende de los resultados de otras clases en C más que de $C_1 = C_i$. Así la gradiente respecto a cada resultado s_i en s solo dependerá de la pérdida dado por el problema binario.

Esta gradiente, respecto de $s_i = s_1$ se escribe como:

$$\frac{\partial}{\partial s_i}(CE(f(s_i))) = t_1(f(s_1) - 1) + (1-t_1)f(s_1) \tag{3.38}$$

Finalmente, esto será escrito como:

$$\frac{\partial}{\partial s_i}(CE(f(s_i))) = \begin{cases} f(s_i) - 1 & \text{si } t_1 = 1 \\ f(s_i) & \text{si } t_1 = 0 \end{cases} \tag{3.39}$$

3.3.7. Optimizador ADAM

La Estimación del Momentum Adaptativo o Adaptative Moment Estimation (ADAM),Kingma (2017), es un método que calcula los ratios de aprendizaje para cada parámetro. Además de almacenar un promedio decreciente exponencialmente de los pasados gradientes cuadrados v_t como en Adadelta (Zeiler (2012)) y RMSprop (Hinton et al. (2017)), Adam también mantiene un promedio exponencialmente decreciente de gradientes pasados m_t, similar al *momentum*.

Calcularemos los promedios decrecientes pasados y los gradientes cuadrados pasados m_t y v_t, respectivamente:

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \end{aligned} \tag{3.40}$$

m_t y v_t son estimados del primer momentum (la media) y el segundo momentum (la varianza no centrada) de las gradientes, respectivamente, por tanto el nombre de este método. Como m_t y v_t son inicializados como vectores de ceros, los autores de Adam observan que ellos están sesgados hacia cero, especialmente durante los pasos iniciales y también cuando los ratios de decrecimiento son pequeños (es decir, β_1 y β_2 se aproximan a 1).

Se corrigen los sesgos en las estimaciones del primer y segundo momentum:

$$\begin{aligned} \tilde{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \tilde{v}_t &= \frac{v_t}{1-\beta_2^t} \end{aligned} \tag{3.41}$$

Luego se usa estas correcciones para actualizar los parámetros, quedando así la regla de actualización de Adam:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\tilde{v}_t} + \epsilon} \tilde{m}_t \tag{3.42}$$

Los autores proponen los valores por defecto para $\beta_1 = 0{,}9$, $\beta_2 = 0{,}999$ y $\epsilon = 10^{-8}$.

3.3.8. Métricas de Clasificación

En un contexto de clasificación binaria, aquí se presentan las principales métricas que son importantes a colocar en orden de evaluar el rendimiento del modelo.

3.3.8.1. Matriz de confusión

La matriz de confusión es usada para tener una vista más completa cuando evaluamos el rendimiento de nuestro modelo. Se define de la siguiente manera:

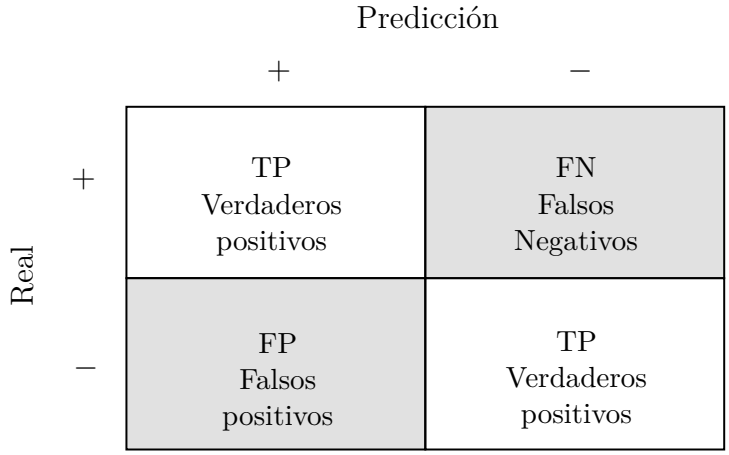


Figura 3.8: Matriz de confusión

3.3.8.2. Principales Métricas

Se organiza la siguiente tabla con las principales métricas comúnmente utilizadas para evaluar el rendimiento de los modelos de clasificación.

Métrica	**Fórmula**	**Descripción**
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Rendimiento general del modelo
Precision	$\frac{TP}{TP+FP}$	Precisión de las predicciones positivas
Recall/Sensitivity	$\frac{TP}{TP+FN}$	Concentración de la muestra positiva actual
Specificity	$\frac{TN}{TN+FP}$	Concentración de la muestra negativa actual
F1 score	$\frac{2TN}{2TP+FP+FN}$	Métrica híbrida ideal para clases no balanceadas.

Cuadro 3.2: Tabla con las métricas más comunes, usadas en la evaluación del modelo.

3.3.8.3. ROC

La curva de operatividad del receptor, también llamada ROC, es la gráfica del TPR contra el FPR variando el umbral. Estas métricas son resumidas en la siguiente tabla:

Métrica	**Fórmula**	**Descripción**
True Positive Rate(TPR)	$\frac{TP}{TP+FN}$	Recall, sensitivity
False Positive Rate(FPR)	$\frac{TN}{TN+FP}$	1-Specificity

Cuadro 3.3: Tabla con las métricas de ROC.

3.3.8.4. AUC

El área bajo la curva de operatividad del receptor, también llamada AUC, es el área bajo la curva ROC como se muestra en la siguiente figura:

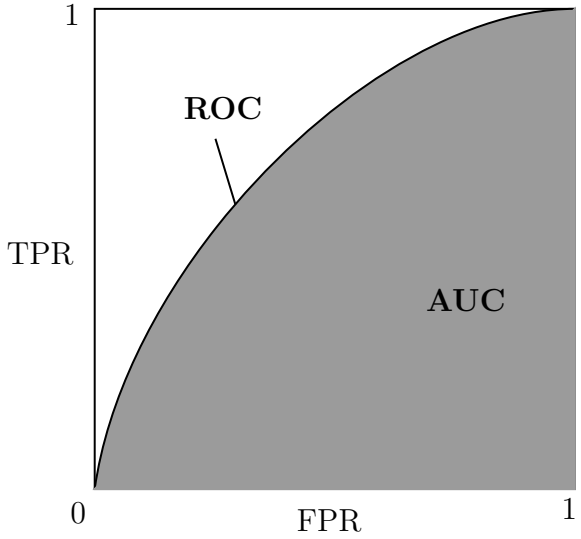


Figura 3.9: AUC: Área bajo la curva

3.4. Importancia

La literatura del Procesamiento de Lenguaje Natural abarca un amplio campo de casos de uso. Aquí estudiamos el enfoque del análisis de sentimientos para datos presentados desde una escritura no formal, por ejemplo, en Twitter las personas se expresan de diferentes maneras y en muchas ocasiones no se respeta la comunicación del lenguaje formal. Para nuestro estudio, nos importa detectar dentro de un conjunto de datos el discurso de odio, es decir, expresiones negativas sobre alguna situación planteada dentro de un contexto de secuencias de palabras. La importancia de nuestro estudio es mostrar la percepción y reproducción del contenido negativo dentro de un medio social, en este caso Twitter, sirviendo un modelo que ensaya el uso de una arquitectura que filtra los datos completos, etiquetados convenientemente, para la detección del odio dentro de los 'tweets'.

En nuestra sociedad, las redes sociales son los medios sociales que más se utilizan, tal como lo describimos en los capítulos introductorios. Por ello es importante se definan reglas para el uso de estas aplicaciones, y así buscar no se fomenten malas actitudes, malos comentarios, información fuera de contexto y contenido malintencionado dentro de esta red.

3.5. Descripción Metodológica

En esta sección mostraremos el paso a paso, en detalle, de nuestro análisis de sentimientos dentro de una data conformada por 6000 *tweets* etiquetados con valores 1 y 0, donde 1 significa que nuestro *tweet* contiene odio, y 0 significa que no.

Primero mencionaremos el origen de los datos y una parte de ellos, luego hablaremos del preprocesamiento realizado a nuestros datos. Seguiremos mostrando parte de nuestro primer análisis a partir de la construcción de nuestro primer modelo Word2Vec, luego mostraremos la arquitectura de nuestro modelo de RNR Bidireccional y por último los resultados obtenidos en el entrenamiento de nuestro modelo.

3.5.1. Datos

Nuestros datos fueron tomados del artículo *Haternet a system for detecting and anlyzing hate speech in Twitter*, Quijano-Sanchez et al. (2019). Tomamos el segundo paquete de datos, el cual está conformado por 6,000 tweets etiquetados mediante una evaluación exhaustiva por los autores de la recolección y clasificación de estos datos.

Los tweets han sido recolectados entre Febrero del 2017 y Diciembre del 2017, obteniendo un tamaño final de 2M de tweets. La naturaleza del idioma de estos tweets es el español de comentarios de personas de España.

	data	label
0	Ismael es egocentrico porque se vuelve loca si le dicen que tiene el pelo bonito se define con otro objetivo #FirstDates251	0
1	..ya tardaba en salir quien pronunciase nombre catalán sílaba aguda como si fuese plana [es Eduááárd],[Ernééést],[Albééért] no son ingleses	0
2	(Esto no es un discurso político y razonado, obviamente, solo una llamada de atención en plan "JODER, NO CUESTA TANTO SABER COSAS")	0
3	Muy despreciados,siiii,pero todos vestidos de alta costura española o extranjera como la derechona burguesa verdad,… https://t.co/5UnIo6C3YF	1
4	marica explicame porque a veces no te entiendo. — En vez de venir a ensuciar mi cc con insultos homof... https://t.co/hSGMfcEtos	1

Cuadro 3.4: Una vista rápida de los datos.

3.5.2. Preprocesamiento de Datos

3.5.2.1. Limpieza de Datos

Nuestros datos son un conglomerado de tweets cuyo contenido está escrito de una manera no formal. Por tanto, es necesario realizar una limpieza de ciertos elementos en la secuencia de datos. Mediante el uso de los paquetes `re`(expresiones regulares) y `nltk` (framework para PLN), realizaremos la limpieza de los datos.

- Se pasan todas las palabras a minúsculas.
- Se eliminan las menciones y url de los datos.
- Se realiza una normalización de las palabras en español, es decir, pasamos valores alfanuméricos y los reemplazamos por letras del alfabeto español. Por ejemplo, las vocales con tildes son reemplazadas con su valor sin tilde.
- En el caso de emojis(iconos o dibujos que representan una emoción dentro del mensaje), se utiliza la librería `emoji` para su tokenización.

- Se eliminan los caracteres especiales, a excepción de algunas letras del español y el símbolo # que está referido a *hashtags*, términos comúnmente usados en los tweets. Más adelante haremos una revisión del impacto de estos elementos en nuestros datos.
- Se eliminan los *stopwords* o palabras vacías del español/castellano. Normalmente son pronombres, preposiciones, adverbios e incluso algunos verbos.

Luego de ejecutar estos pasos, logramos obtener los datos limpios:

id	data
0	ismael egocentrico vuelve loca dicen pelo bonito face_with_tears_of_joy face_with_tears_of_joy face_with_tears_of_joy face_with_tears_of_joy defin...
1	tardaba salir pronunciase nombre catalan silaba aguda plana eduaaard erneeest albeeert ingleses
2	discurso politico razonado obviamente solo llamada atencion plan joder cuesta saber cosas
3	despreciados siiii vestidos alta costura española extranjera derechona burguesa verdad
4	marica explicame veces entiendo vez venir ensuciar insultos homof

3.5.2.2. Tokenización

Para nuestro caso, la tokenización de los datos sirve para organizar las palabras como secuencias de palabras, esta representación sirve para su posterior uso en los modelos de *word embeddings* a ser entrenados.

id	data
0	[ismael, egocentrico, vuelve, loca, dicen, pelo, bonito, face_with_tears_of_joy, face_with_tears_of_joy, face_with_tears_of_joy, face_with_tears_o...
1	[tardaba, salir, pronunciase, nombre, catalan, silaba, aguda, plana, eduaaard, erneeest, albeeert, ingleses]
2	[discurso, politico, razonado, obviamente, solo, llamada, atencion, plan, joder, cuesta, saber, cosas]
3	[despreciados, siiii, vestidos, alta, costura, española, extranjera, derechona, burguesa, verdad]
4	[marica, explicame, veces, entiendo, vez, venir, ensuciar, insultos, homof]

3.5.2.3. Stemming o Radicalización

Se llama *stemming* al procedimiento de convertir palbras en raíces. Estas raíces son la parte invariable de palabras relacionadas sobre todo por su forma. Una de sus ventajas es que puede relacionar palabras de distinta clase. Por ejemplo *ordenar* y *orden* tienen la misma raíz *orde-*. Por tanto el *stemming* puede reducir el númeo de elementos que forman nuestros textos.

id	data
0	[ismael, egocentrico, vuelv, loca, dicen, pelo, bonito, face_with_tears_of_joy, face_with_tears_of_joy, face_with_tears_of_joy, face_with_tears_of...
1	[tardaba, salir, pronuncias, nombr, catalan, silaba, aguda, plana, eduaaard, erneeest, albeeert, ingles]
2	[discurso, politico, razonado, obviament, solo, llamada, atencion, plan, joder, cuesta, saber, cosa]
3	[despreciado, siiii, vestido, alta, costura, española, extranjera, derechona, burguesa, verdad]
4	[marica, explicam, vece, entiendo, vez, venir, ensuciar, insulto, homof]

3.5.2.4. Importancia de los *Hashtags*

Los *hashtags*, son etiquetas o palabras clave adjuntas en algunos tweets que a lo largo del *corpus* están presentes. Por tal motivo, es necesario mantener esta información dentro de los datos, pues sería otro indicador que mida la intención de un tweer. Graficamos la distribución de los *hashtags* en nuestro corpus, tanto para los tweets positivos y negativos.

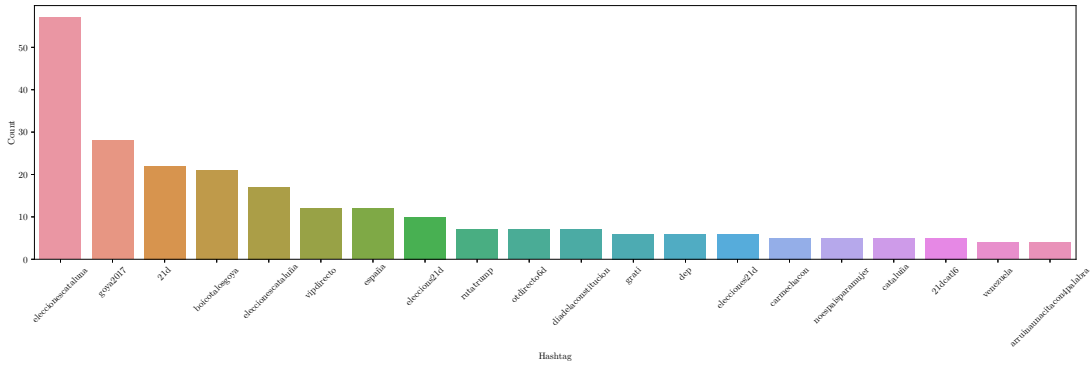


Figura 3.10: Histograma de la distribución de los *hashtags* positivos en el corpus

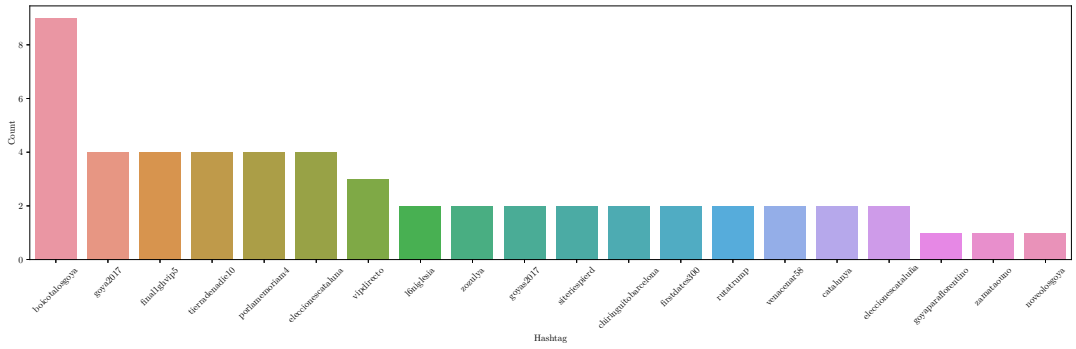


Figura 3.11: Histograma de la distribución de los *hashtags* negativos en el corpus

3.5.3. Implementación de Word2Vec

En este apartado veremos la metodología de la implementación de Word2Vec para la generación de nuestros *embeddings* de nuestro corpus. Nuestros datos de entrada serán los datos preprocesados mediante los conceptos previos. Para la construcción de nuestro modelo Word2Vec haremos el uso de la librería Gensim, Rehurek and Sojka (2011); esta librería puedo desarrollar el entrenamiento de forma iterativa de Word2Vec, creando un diccionario de distintas palabras usadas en nuestro *corpus*.

Acontinuación la configuración de nuestro modelo:

Parámetro	Valor
Tamaño del embedding/$N°$ de características independientes	50
Tamaño del contexto o *window*	5
Frecuencia mínima por palabra	3
Skip-Gram habilitado	1
Número de Negative Sampling	10
Epochs o número de iteraciones	10

Cuadro 3.5: Configuración del modelo Word2Vec con Gensim.

Según Jurafsky (2020), el tamaño de los *embeddings* es importante para evitar el *overfitting* de nuestro resultados, por ello utilizaremos valores de 50,100,200 y 300 para nuestro análisis.

3.5.4. Implementación del modelo RNR Bidireccional - LSTM(BiRNR-LSTM)

3.5.4.1. Arquitectura del modelo

Para el diseño de nuestra arquitectura, nos hemos basado del modelo utilizado en el artículo, Ozan (2020). En orden a implementar nuestra arquitectura, utilizaremos los paquetes de Tensorflow, Abadi et al. (2015) y Keras, Chollet et al. (2015).

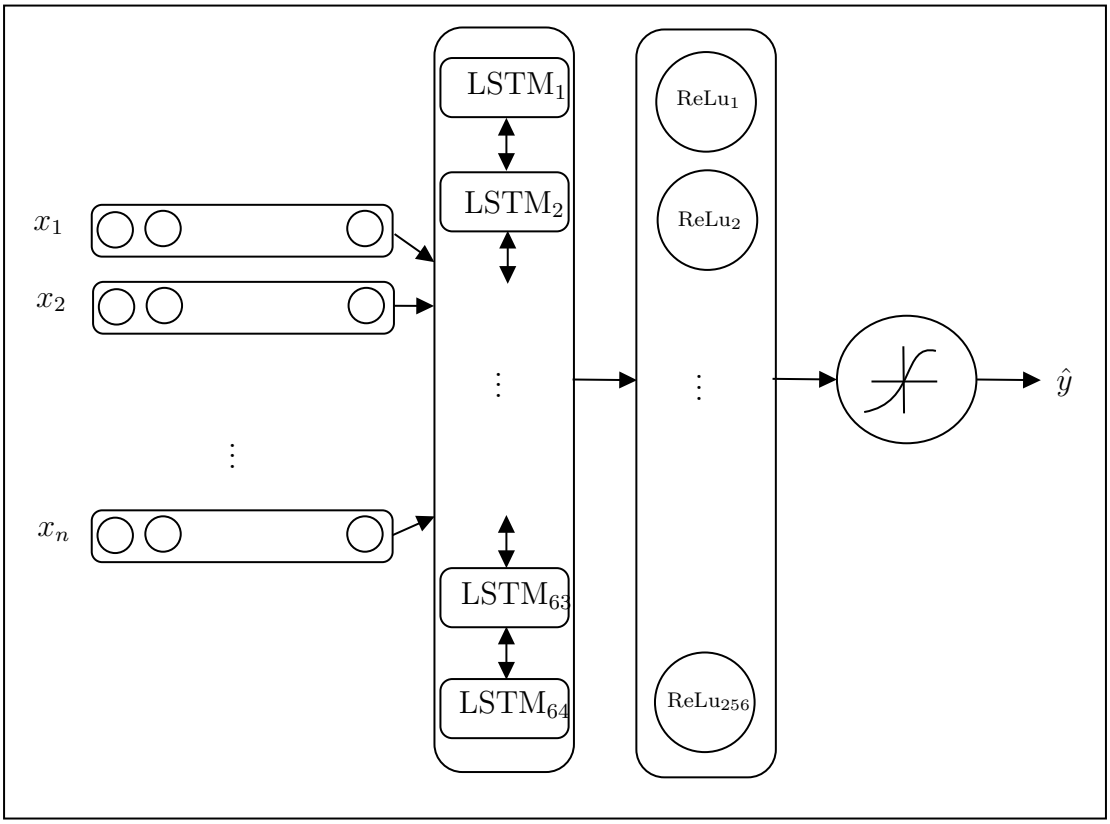


Figura 3.12: Una vista de la arquitectura de nuestro modelo a implementar.

Detallaremos las fases de nuestro modelo:

1. Capa de entrada: Se utilizará una capa de entrada conformada por los *embed-*

dings de las palabras de nuestro vocabulario. El tamaño de nuestra matriz de entrenamiento será de dimensión $(11652, n)$, donde n es el tamaño del *embedding* y 11652 es el número de palabras en nuestro vocabulario. Como sabemos no todas nuestros tweets o datos contienen el mismo número de palabras, por tanto se debe aplicar la técnica del *padding*, por ejemplo si tomamos $n = 100$, las secuencias de palabras que sean menor a esa leongitud serán rellenadas con 0, y las que superen esa longitud, serán concatenadas hasta lograr el tamaño debido.

2. La capa BiRNR-LSTM: Este es el eje de nuestro modelo, ya que aquí se concentrará el mayor ajuste de nuestro modelo. Se utilizará en esta capa 64 nodos LSTM.

3. La capa oculta densa: Se utiliza la capa oculta densa completamente conectada a todos los nodos LSTM de la capa BiRNR-LSTM. Se utilizarán 256 nodos cuyas salidas serán evaluadas con la función de activación ReLu (tanh).

4. La capa de salida está compuesta por un único nodo sigmoide, ya que este es un problema de clasificación binaria.

5. La función de pérdida será la Binary Cross-Entropy (BCE), que medirá todas las diferencias entre las etiquetas predecidas $\hat{y}$ y la etiqueta de verdad o objetivo y.

6. Por último se utilizará el optimizador ADAM para las correcciones en la fase de entrenamiento.

3.6. Resultados

En esta sección mostraremos los resultados experimentales de nuestro análisis de sentimientos. Primero mostraremos algunos resultados de nuestro modelo Word2Vec, usando Skip-Gram con Negative Sampling. Básicamente se hace uso del método $most_similar()$, provisto por Gensim, que encuentra la similaridad de las palabras del corpus(contexto), con la palabra que queremos aproximar.

En la siguiente tabla se pueden ver algunas aproximaciones de este modelo:

Palabra entrada	Palabras Similares	Ratio de Similitud(SR)
dictador	maduro	0.9355
	franco	0.8736
	venezuela	0.8643
	genocida	0.8503
	dictadura	0.8305
mamon	mamona	0.9571
	mongolo	0.9510
	evil_monkey (🙈)	0.9407
	lamentable	0.9407
	face-with-tears-of-joy (😂)	0.9356
fascista	fascismo	0.9055
	concentracion	0.8990
	ucraniano	0.8893
	corrupto	0.8790
	neonazi	0.8782
españa	español	0.9182
	#boicotalosgoya	0.9166
	cine	0.9098
	spain	0.9058
	patriotismo	0.8959
racismo	ataque	0.9852
	agresión	0.9795
	medida	0.9776
	aceptar	0.9776
	comparar	0.9769

Cuadro 3.6: Resultados del modelo Word2Vec con Negative Sampling.

Podemos inferir de estos resultados que nuestro modelo logra una precisión alta respecto a la predicción del contexto para nuestra palabras de entrada. Podemos observar en la tabla que incluso es posible relacionar palabras con representaciones visuales como 'emojis' y 'hashtags', haciendo de este modelo robusto.

Para el caso de la RNR, se ha entrenado la arquitectura definida en la figura 3.12, para 4 diferentes tamaños de *embeddings*: 50,100,2000 y 300. Nuestra capa de entrada recibirá como *input* la matriz de *embeddings* entrenada con nuestro modelo de Word2Vec con Negative Sampling, para cada entrenamiento hemos pasado por 100 *epochs* o iteraciones del modelo. Se ha utilizado una segmentación de la data de $90\,\% - 10\,\%$ para los datos de entrenamiento y validación, respectivamente. A continuación mostraremos la gráfica 3.13 de la función de pérdida o *loss* de la fase de entrenamiento y validación para cada tamaño de *embedding*.

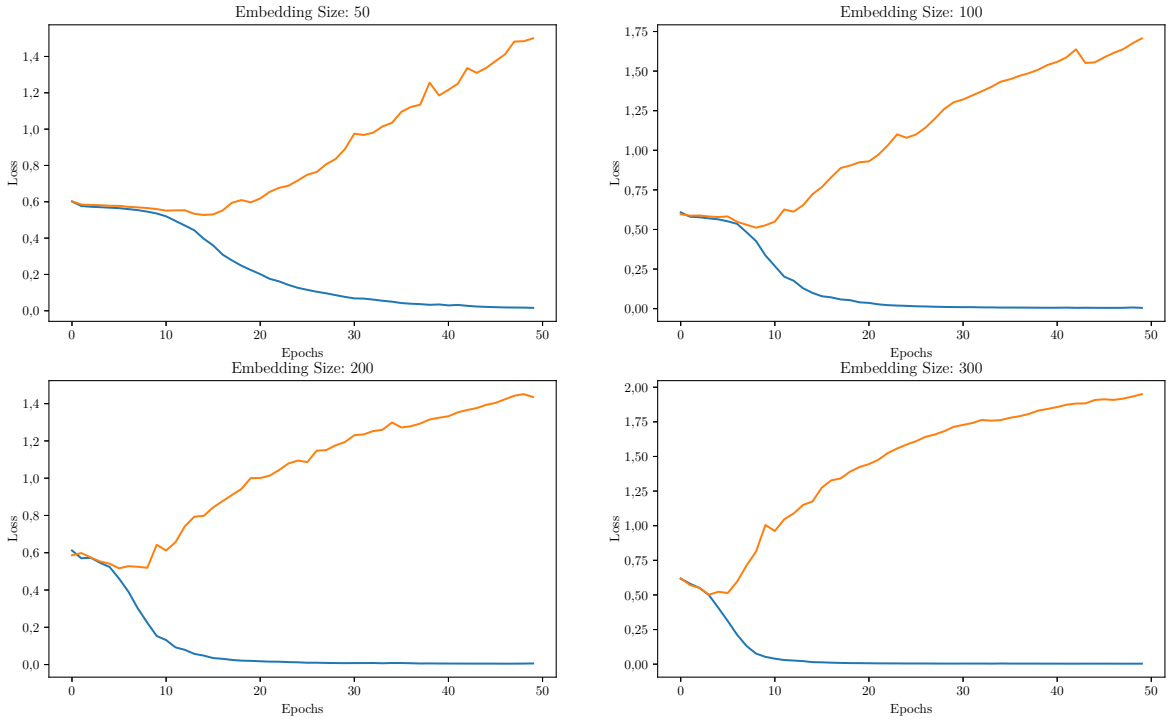


Figura 3.13: Gráficas del error en las fases de entrenamiento y validación del modelo para los *embeddings* de tamaño 50,100,200 y 300, respectivamente. Cada experimento fue entrenado por 100 *epochs*

El problema recurrente en estos gráficos es que el error en la validación crece y decrece en el mismo punto (mínimo global). Este incremento es un efecto del problema *overfitting* o sobreajuste de los parámetros en el modelo. Graficaremos las gráficas del error de validación (Validation Loss) para cada tamaño de *embedding* en una sola figura.

En la figura 3.14, podemos observar que el error decrece, y tan pronto alcanza su mínimo global, empieza a incrementar. Podemos ver que los mínimos son muy cercanos para ciertos tamaños de *embedding*, sin embargo difieren en el *epoch* en el que alcanzan este valor. Se puede observar, además, que el error en la validación empieza a incrementa en el *epoch* 14 correspondiente a los *embeddings* de tamaño 50 y en el *epoch* 3 correspondiente a los *embeddings* de tamaño 300. De aquí podemos decir que para tamaños más altos de un *embedding*, este comenzará a hacer *'overfitting'* en los primeros *epochs*. Sin embargo aún tenemos dudas sobre que tamaño de *embedding* debemos utilizar.

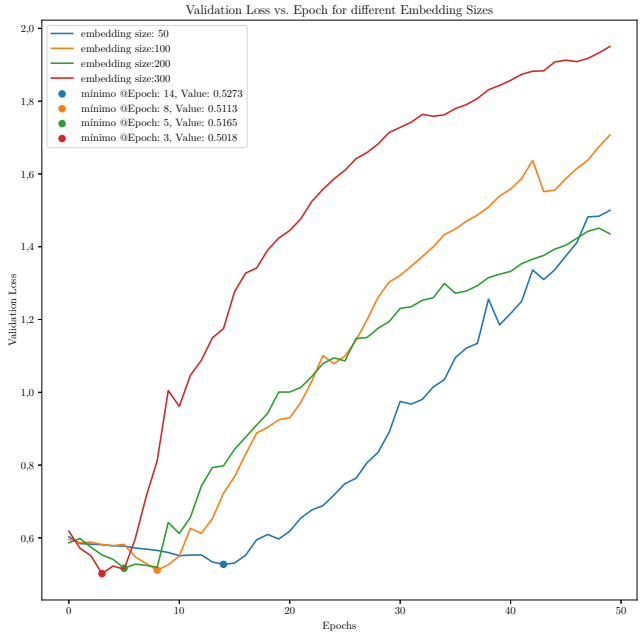


Figura 3.14: Error en las fases de validación del modelo para los *embeddings* de tamaño 50,100,200 y 300, graficados juntos. Los puntos indican donde el error en la validación alcanza su valor mínimo global.

Para decidir ello, echaremos una vista a la gráfica de la métrica F1-Score con respecto a los *epochs*, F1-Score es una buena métrica para problemas de clasificación binaria desde que este captura la precisión y los ratios de exhaustividad (Recall) en un solo valor.

De la figura 3.15, vemos que se grafican los puntos encontrados en la gráfica del error de validación de cada *embedding* visto en la 3.14. Es en estos puntos donde el modelo empieza a hacer el *overfitting*. Estos puntos son importantes para decidir en que punto realizaremos una pausa prematura o *early stopping*, Yao et al. (2007), desde que el objetivo principal de nuestro modelo es minimizar el error usando la gradiente descendiente, utilizar esta herramienta es muy útil para mejorar el rendimiento del modelo. Los puntos en la gráfica nos indica en qué punto empezará a hacer el *overfitting* nuestro modelo. Tenemos que en los *epochs* 14, 8, 5 y 3 para los tamaños de *embedding* 50, 100, 200 y 300, respectivamente, el modelo comenzará a hacer el *overfitting*. Está claro que realizar mayores pasos en el entrenamiento asegurará una mayor precisión en nuestro modelo, por tanto mediante esta idea, tomaremos el tamaño de *embedding* 50 como el que mejor se portará dentro de nuestro modelo.

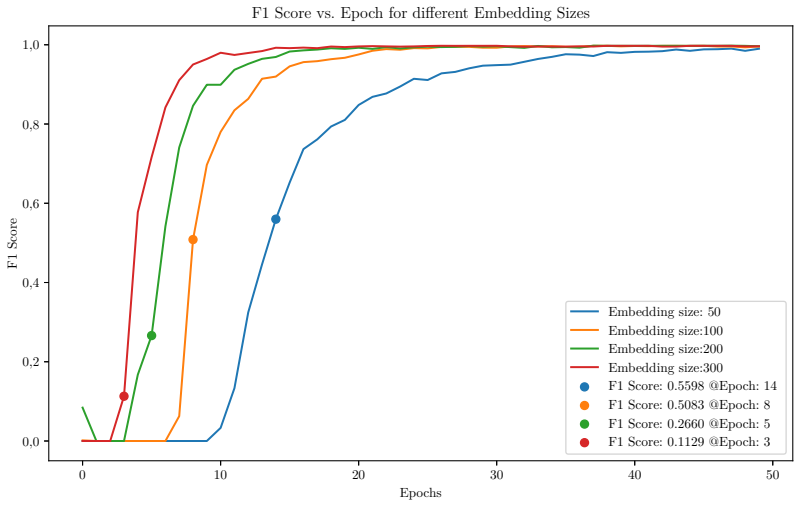


Figura 3.15: F1-Score basado en el entrenamiento de la 3.14. Los puntos indica los *epochs* donde el sistema comienza a hacer el *overfitting*.

En la siguiente tabla colocaremos los resultados de nuestro modelo para cada tamaño de *embedding*:

Tamaño de Embedding	Epoch(Mínimo global)	Accuracy	F1-Score	Loss
50	14	69.83 %	0.5598	0.5273
100	8	70.99 %	0.5083	0.5113
200	5	73.00&	0.2060	0.5165
300	3	75.49&	0.1129	0.5018

Cuadro 3.7: Resultados Generales del Modelo BiRNN-LSTM

Capítulo 4

Conclusiones

En la literatura, hemos visto que los resultados del uso de nuestros modelos tienen una buena aproximación con respecto a los valores verdaderos u objetivos. El análisis de sentimientos realizado buscó encontrar los patrones en el que un tweet pueda contener odio o no. Para ello se construyeron 2 modelos, utilizados en la literatura del Procesamiento de Lenguaje Natural: Word2Vec con Negative Sampling y un modelo de clasificación binaria usando una RNR Bidireccional-LSTM.

En el primer estudio, hicimos el uso de la librería Gensim para poder construir nuestro modelo y encontrar los ratios de similaridad de las palabras de entrada con su contexto, se dio cuenta de la calidad del modelo, el cual fue entrenado con 6000 tweets o datos etiquetados tomados de un artículo previo. Se observo que nuestro modelo tiene la capacidad de incluso obtener similitudes entre palabras del diccionario con representaciones visuales como 'emojis' y 'hashtags'.

En el siguiente modelo, se entreno una arquitectura cuyo núcleo principal esta en la capa Bidireccional con unidades LSTM. Tomando los *embeddings* entrenados del modelo de Word2Vec con Negative Sampling. Se entrenaron 4 casos para tamaños de *embeddings* 50,100,200 y 300. De acuerdo a nuestros resultados, concluimos que para un tamaño de *embedding* igual a 50, es el tamaño que logra mejores resultados en nuestro modelo. Se dió cuenta también de la noción de *early stopping*, el cual es una estrategia muy buena para decidir en que momento nuestro modelo debe parar para evitar el *overfitting*. Además que esta estrategia evita la saturación de los recursos del hardware utilizado para el entrenamiento.

Capítulo 5

Recomendaciones

Se vio que nuestro modelo logra una aproximación o precisión buena, en breve, con un *accuracy* del 70 % aproximadamente. Es posible entonces para futuros trabajos, realizar una evaluación más exhaustiva del preprocesamiento de nuestros datos, se debería hacer más experimentos para distintos tamaños de *embeddings* y observar si para algún valor mayor o menor que 50, nuestro modelo puede lograr mejores resultados. También es posible dar cuenta que el modelo Bidireccional-RNR-LSTM es muy útil para problemas de clasificación, podríamos ensayar otros modelos para poder encontrar un modelo mucho más óptimo, cuyos resultados en la predicción pueda alcanzar un valor aún más real en los datos.

Bibliografía

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Alsafari, S., Sadaoui, S., and Mouhoub, M. (2020). Effect of word embedding models on hate and offensive speech detection.

Bashir, D., Montanez, G. D., Sehra, S., Segura, P. S., and Lauw, J. (2020). An information-theoretic perspective on overfitting and underfitting.

Buscaldi, D., Hernández-Farias, I., and Ferrández (2015). Sentiment analysis on microblogs for natural disasters management. *Proceedings of the 24th International Conference on World Wide Web - WWW '15 Companion*.

Chaubard, F., Fang, M., Genthial, G., Mundra, R., and Socher, R. (2019). Lecture notes: Part i. word vectors i: Introduction, svd and word2vec. *CS224n: Natural Language Processing with Deep Learning*.

Chauhan, P., Sharma, N., and Sikka, G. (2021). The emergence of social media data and sentiment analysis in election prediction. *Journal of Ambient Intelligence and Humanized Computing*, 12.

Chollet, F. et al. (2015). Keras.

Christos Troussas, M. V. (2020). *Advances in Social Networking-based Learning*. Springer International Publishing, 1 edition.

Davidson, T., Warmsley, D., Macy, M., and Weber, I. (2017). Automated hate speech detection and the problem of offensive language. *Proceedings of the International AAAI Conference on Web and Social Media*, 11(1):512–515.

Del Vigna, F., Cimino, A., Dell'Orletta, F., Petrocchi, M., and Tesconi, M. (2017). Hate me, hate me not: Hate speech detection on facebook. *First Italian Conference on Cybersecurity (ITASEC17)*.

Hinton, G., Srivastava, N., and Swersky, K. (2017). Lecture 6a: Overview of mini-batch gradient descent. *Neural Networks for Machine Learning*.

Irsoy, O. (2013). Bidirectional recursive neural networks for token-level labeling with structure. *CoRR*, abs/1312.0493.

Jiang, L. and Suzuki, Y. (2019). Detecting hate speech from tweets for sentiment analysis. *2019 6th International Conference on Systems and Informatics (ICSAI)*, pages 671–676.

Jurafsky, D. (2020). *Speech and Language Processing*. Stanford University, 3 edition.

Kauffmann, E., Peral, J., Gil, D., Ferrández, A., Sellers, R., and Mora, H. (2019). Managing marketing decision-making with sentiment analysis: An evaluation of the main product features using text data mining. *Sustainability*, 11(15).

Kingma, D. P. (2017). Adam: A method for stochastic optimization.

Kozubek, T., Blaheta, R., Šístek, J., Rozložník, M., and Čermák, M. (2016). *High Performance Computing in Science and Engineering: Second International Conference, HPCSE 2015, Soláň, Czech Republic, May 25-28, 2015, Revised Selected Papers*. Lecture Notes in Computer Science. Springer International Publishing.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality.

Monica Anderson, B. A. (2021). Social media use in 2021. https://www.pewresearch.org/internet/2021/04/07/social-media-use-in-2021/.

Ozan, S. (2020). Case studies on using natural language processing techniques in customer relationship management software. *Journal of Intelligent Information Systems*, 56(2):233–253.

Quijano-Sanchez, L., Kohatsu, J. C. P., Liberatore, F., and Camacho-Collados, M. (2019). Haternet a system for detecting and analyzing hate speech in twitter.

Rehurek, R. and Sojka, P. (2011). Gensim–python framework for vector space modelling. *NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic*, 3(2).

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). *Learning Representations by Back-Propagating Errors*, page 696–699. MIT Press, Cambridge, MA, USA.

Twitter (2021a). Q1 2021, letter to shareholders. *Twitter Investor Relations*.

Twitter (2021b). The twitter rules. https://help.twitter.com/en/rules-and-policies/twitter-rules.

Yao, Y., Rosasco, L., and Caponnetto, A. (2007). On early stopping in gradient descent learning. *Constructive Approximation*, 26(2):289–315.

Zeiler, M. D. (2012). Adadelta: An adaptive learning rate method.