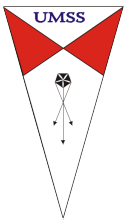

UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
CARRERA DE INGENIERÍA INFORMÁTICA

LAS REDES NEURONALES CONVOLUCIONALES EN EL ANÁLISIS DE SENTIMIENTOS DE LAS OPINIONES DE ATRACTIVOS TURÍSTICOS DE BOLIVIA

Tesis, Presentada Para Optar al Diploma Académico de Licenciatura en Ingeniería Informática.

Presentado por: Diego Alexander García Cuchallo
Tutor: Msc. Lic. Patricia Elizabeth Romero Rodríguez

Cochabamba - Bolivia
Marzo, 2019

Dedicatoria

Dedico este trabajo a mis padres y hermanos, por su apoyo constante, por sus valiosos consejos y por siempre estar presentes.

Agradecimientos

Mi agradecimiento a todos, mi tutora, mi familia, mis amigos que de una u otra manera me brindaron su colaboración y se involucraron en este proyecto.

FICHA RESUMEN

El auge de las redes sociales ha popularizado la difusión de opiniones en la web, donde uno no solo es consumidor sino creador. Esto ha llamado la atención de los negocios que viven de la opinión de sus clientes, por lo que surgió la necesidad de clasificar opiniones por polaridad y así mejorar sus servicios con la retroalimentación de los usuarios.

Abrumados por la cantidad de comentarios y el laborioso trabajo que significa realizar una clasificación manual, los sistemas automatizados de clasificación de textos se vuelven una necesidad. La meta es etiquetar textos escritos en lenguaje natural según la polaridad que estos expresan, ya sea negativa, positiva o neutral.

La clasificación automática de opiniones es un área emergente que se conoce como "Análisis de sentimientos", una tarea de clasificación en procesamiento de lenguaje natural. A lo largo del tiempo este problema ha sido frecuentemente enfrentado con métodos de clasificación de *Machine learning* como Regresión logística o *SVM* (*Support Vector Machine*).

Esta investigación planteó una solución novedosa para el entorno de las opiniones turísticas en Bolivia. Las redes neuronales convolucionales, una técnica de *Deep learning* (subárea de *Machine learning*) ampliamente usada para reconocimiento de imágenes, fueron usadas para clasificar opiniones escritas en lenguaje natural.

Debido a que no existía un conjunto de datos publicado utilizable para los propósitos de este trabajo, se desarrolló un conjunto propio con opiniones esparcidas en la web. Adicionalmente se planteó un número de propiedades que ayudarían a entender la calidad de estos datos.

Los esfuerzos se centraron en diseñar un modelo clasificador de 3 etiquetas, estos son Positivo, Negativo y Neutral. Experimentando con los parámetros que determinan la composición de un modelo de red neuronal convolucional se obtuvieron resultados muy similares a los del estado del arte.

Se realizó una comparación entre diferentes diseños de redes neuronales convolucionales, con tal de encontrar algún patrón que indique cómo se puede mejorar aún más la precisión. Se llegó a la conclusión que el conjunto de datos conformado no es lo suficientemente grande para un diseño de red neuronal convolucional muy complejo pero es suficiente para probar la hipótesis de esta investigación.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ALGORITMOS

CAPÍTULO 1 INTRODUCCIÓN

En los últimos años el mundo se ha visto inmerso en la revolución de las redes sociales, donde por primera vez se vió una gran cantidad de opiniones almacenadas de manera digital y listas para el procesamiento, resultando en aplicaciones tales como: Recomendación de productos, predicción de resultados, negocios inteligentes, etc.

Estas aplicaciones utilizan algoritmos, técnicas y herramientas de la ciencia de computación con la finalidad de clasificar textos según la polaridad de una opinión, tal proceso es conocido como **análisis de sentimientos**.

Con el tiempo la exactitud ha mejorado y nuevas técnicas siguen apareciendo, últimamente muchos de los avances pertenecen al área de *Deep learning*[1] obteniendo resultados bastante aceptables.

Cuando se aborda un problema de análisis de sentimientos, la primera preocupación es determinar qué tipo de características[2] se utilizará. Este proceso es empírico ya que existen variedad de técnicas y cada una tiene sus peculiaridades, esta incertidumbre hace que el análisis de sentimientos sea un arte.

La novedad que aporta *Deep learning* está en evitar seleccionar las características manualmente. En su lugar estos se aprenden automáticamente utilizando técnicas de aprendizaje no supervisado[3]. Resultado de aprender las características se obtienen vectores multidimensionales representativos creados a partir de los datos crudos. Aquí es donde brillan las Redes Neuronales Convolucionales (*RCN*) ya que se especializan en manejar características con alta dimensionalidad.

[1] Aprendizaje en profundidad, es una subárea de *Machine learning* que explota la utilización de redes neuronales con más de una capa escondida.

[2] Atributo de los datos, se refiere una característica (o columna si son datos estructurados) de los datos propicio para conformar un modelo.

[3] Tipo de aprendizaje donde el objetivo es encontrar similitudes entre los datos para formar agrupaciones.

1.1. ESTADO DEL ARTE

El objeto de estudio del presente proyecto es el análisis de sentimientos, también llamado minería de opiniones, es una subárea del procesamiento de lenguaje natural el cual tiene muchos retos aún en la actualidad (Liu, 2012).

Desde el año 2000 este campo tiene un gran impulso gracias a la proliferación de opiniones en la web debido a las redes sociales y el comercio electrónico. Las investigaciones abordan este problema desde diferentes puntos de vista. Basándose en fuentes de datos como blogs, reseñas, bancos de datos o *tweets*[4].

Los algoritmos más utilizados para el análisis de sentimientos se encuentran en la rama de *Machine learning* (Aprendizaje de la máquina). Algoritmos como *Naïve Bayes* han sido utilizados para clasificar opiniones desde las primeras investigaciones (Vyrva, 2016), otros conocidos y entre los que mejores resultados presentan son *SVM* (*Support Vector Machine*) y las redes neuronales clásicas.

En el ámbito de redes neuronales más complejas como las de convolución también hay bastantes avances, investigaciones recientes como la de Kim (2014) que plantea una arquitectura de red convolucional sencilla para realizar tareas de clasificación. También hay propuestas más complejas como Hughes, Li, Kotoulas, y Suzumura (2017) que fue aplicado al problema de clasificación de documentos médicos con 16 clases diferentes.

En aproximaciones más antiguas se tiene a Collobert y cols. (2011) que utiliza una capa de convolución para aprender vectores de palabras, seguido de capas de completamente conectadas para realizar clasificación.

En años más recientes se tiene el *framework* de red neuronal convolucional para análisis de sentimientos propuesto por Dos Santos y Gatti (2014). Este enfoque descuartiza y analiza los textos empezando por cada carácter hasta llegar a las oraciones.

1.2. PREGUNTA DE INVESTIGACIÓN E HIPÓTESIS

Debido a la variación y ambigüedad de las opiniones escritas en lenguaje natural, se dificulta la automatización de la clasificación de textos. Se ha considerado a las *RCNs* como técnica interesante por lo que se plantea la siguiente pregunta.

Pregunta: ¿Cuáles son los parámetros de las redes neuronales convolucionales que

[4]Mensajes cortos particulares de la red social de *Twitter*.

influyen en la exactitud?

Hipótesis: La calidad de los datos de entrenamiento y el diseño de la red neuronal convolucional.

1.3. OBJETIVOS

Objetivo general: Evaluar el grado de acierto de las redes neuronales convolucionales en la clasificación de opiniones escritas en lenguaje natural.

Objetivos específicos:

1. Modelar la red neuronal convolucional en base a las características del caso de estudio, opiniones de los atractivos turísticos de Bolivia.
2. Implementar el modelo de red neuronal convolucional para el caso de estudio.
3. Establecer las características de los datos de aprendizaje y de pruebas, para el caso de estudio.
4. Experimentar con el modelo de red neuronal convolucional a fin de identificar las condiciones bajo las cuales los resultados son los esperados.

1.4. JUSTIFICACIÓN

El análisis de sentimientos, ofrece amplias posibilidades de ser aplicado, sobre todo en el comercio electrónico. En este proyecto se pretende comprobar que las redes neuronales convolucionales dedicadas a clasificación de sentimientos obtienen resultados aceptables, acorde a las necesidades del entorno.

Las *RCN* son una técnica relativamente nueva en el área del procesamiento de lenguaje natural, aunque existen investigaciones aplicadas al análisis de sentimientos, si se refiere al entorno boliviano no existen implementaciones de este tipo.

El desarrollo de esta red provee de un clasificador de opiniones que puede ser utilizado en un contexto boliviano, significando en un aporte académico experimental.

Por el lado aplicativo, se ha elegido como caso de estudio las opiniones en el idioma español sobre los atractivos turísticos que se encuentran en Bolivia. Es conveniente aclarar

al respecto que si bien es cierto existen librerías disponibles para el procesamiento de lenguaje natural, estas tienen la limitante de funcionar para el idioma inglés. Por lo que la construcción y entrenamiento de una red neuronal convolucional para el idioma español es un aporte interesante que podría abrir futuras investigaciones y aplicaciones.

1.5. ALCANCES Y LÍMITES

La principal área del proyecto es la Inteligencia Artificial, especializándose en el campo de *Deep learning*. Como se plantea el proyecto los esfuerzos se centrarán en el desarrollo de una *RCN* aplicada al problema de análisis de sentimientos.

Debido al factor de innovación en la aplicación de las *RCN* para la clasificación de opiniones, se debe aclarar que el software obtenido es de tipo experimental, con la funcionalidad básica y necesaria para realizar la verificación de la hipótesis formulada en párrafos anteriores.

Los textos para el aprendizaje de la *RCN*, fueron extraídos del dominio turístico, es decir, comentarios de personas que vierten sus opiniones en idioma español hacia atractivos turísticos de Bolivia.

1.6. ESTRUCTURA GENERAL DEL DOCUMENTO

Este documento está dividido por capítulos que deberían ser abordados en orden para una mejor compresión lectora de su contenido, aunque si el lector ya cuenta con conocimientos básicos de los fundamentos de esta investigación como son las ramas de *Machine learning* y Análisis de sentimientos se puede saltar directamente al Capítulo 3.

En el Capítulo 2 se tocarán los temas que sustentan esta investigación, que son los mismos mencionados en el párrafo anterior. Posteriormente se relata el desarrollo y ejecución de la investigación, resumido en los puntos explicados a continuación:

- Una primera etapa se cubre en el Capítulo 3 donde se explica la recolección de los datos, criterios de selección, balance, tamaño y representatividad de los mismos. Además del procesamiento que se realiza para que los textos sean una entrada propicia para redes neuronal.
- El Capítulo 4 procederá a explicar el diseño de la red neuronal de convolución con

detalles conceptuales específicos de esta, además de las herramientas usadas para la implementación.

- En el Capítulo 5, se mostrará una pequeña aplicación construida sobre la red antes entrenada y los experimentos realizados que determinaron la configuración y diseño de red propicios.
- Finalmente el Capítulo 6, el último, presentará las conclusiones de la investigación de acuerdo a los resultados mostrados en el Capítulo 5 y además los trabajos futuros que pueden surgir a partir de este trabajo.

CAPÍTULO 2 MARCO TEÓRICO

En este capítulo se tocarán los principales conceptos que soportan esta investigación, Análisis de sentimientos y *Machine learning*. Posteriormente se introducirá a una de las áreas de inteligencia artificial más influyentes en los últimos años *Deep learning*. Se resumirán los conceptos base e introducirá la terminología a usar en este documento.

El tema que más concierne al desarrollo de la hipótesis de esta investigación, las redes neuronales convolucionales, se verán un poco más en detalle. Las secciones de más adelante suponen que el lector tiene conocimientos fundamentales sobre álgebra lineal y cálculo, necesarios para comprender el contenido de este y los siguientes capítulos.

2.1. ANÁLISIS DE SENTIMIENTOS

El análisis de sentimientos es un subárea del procesamiento de lenguaje natural que estudia la polaridad de la opinión que tiene un sujeto sobre un objetivo. Según Liu (2012) es el campo que analiza las opiniones, evaluaciones, actitudes y emociones de las personas hacia entidades tales como productos, organizaciones, eventos, temas y sus atributos.

La literatura suele usar los términos análisis de sentimientos o minería de opiniones indistintamente, en este documento se referirá a este problema como análisis de sentimientos.

2.1.1. Aplicaciones

El término análisis de sentimientos apareció por primera vez en 2003, sin embargo han existido investigaciones al respecto desde el año 2000.

La utilidad comercial es una de las razones del gran avance que ha tenido este campo. Con el tiempo la industria encontró numerosos usos y hoy en día el análisis de sentimientos tiene cabida en casi cualquier rubro, además de ser utilizada en aplicaciones comerciales.

El acelerado progreso que ha tenido esta área en los últimos años se debe al auge de las redes sociales. Este fenómeno ha logrado la formación de bancos de datos colosales a velocidades exorbitantes, información que se acumula y aprovecha para alimentar sistemas tales como los clasificadores de opiniones. A continuación se describen algunos usos del análisis de sentimientos:

Negocios inteligentes

La opinión de los consumidores indica la aceptación que posee un producto en el mercado, estos datos son importantes para las empresas ya que pueden predecir si un producto similar tendrá el mismo éxito o no. Los sistemas de análisis de sentimientos abordan esta preocupación utilizando estrategias automatizadas y devuelven resultados sin necesidad de intervención humana. Por ejemplo, en el negocio del cine existen sistemas que se alimentan de reseñas de películas para predecir la cantidad de dinero que recaudará un título en su primera semana en cartelera.

En la toma de decisiones

Suponga el caso de una persona que quiere comprar un producto de calidad, en el pasado está persona habría buscado la opinión de familiares o amigos para tener un punto de referencia. En la actualidad eso ya no es necesario, porque es posible encontrar reseñas, usos, ventajas, desventajas y hasta consejos sobre dicho artículo en la web. Pero una persona corriente no tiene tiempo para leer todas las reseñas, ahí es donde entran los sistemas de análisis de sentimientos que se alimentan de todas las opiniones concernientes para mostrar información resumida sobre un producto de interés. El mismo sistema puede aplicarse a casos como: Elecciones de presidente, películas, o restaurantes.

Anuncios en la web

La publicidad en la web ahora es inteligente, el navegador recuerda información del usuario como por ejemplo acciones e intereses. Con estos datos los sitios web son capaces de determinar qué anuncio es más apropiado mostrar. Cuando uno se dispone a comprar algo en un sitio web, propaganda similar a anteriores compras es mostrada. Por otro lado, si la mercancía es de desagrado se muestran unas diferentes o de otra marca. Esto es un programa de recomendación haciendo su trabajo.

Spam de opiniones

Se puede notar que las opiniones tienen gran influencia en la población, por este motivo han surgido individuos que tratan de tomar ventaja de esto creando opiniones falsas, ya sean positivas o negativas según sea el objetivo del individuo. Esta actividad es conocida como **Spam de opiniones** y suele ser utilizada para publicidad o incluso ataques.

Debido a que los portales de opinión y las redes sociales deben continuar siendo una fuente confiable, esta actividad tiene que ser detectada. El problema es que no es como otros tipos de spams, y es muy difícil reconocer si una opinión es o no legítima.

2.1.2. Niveles de análisis

Las investigaciones en el análisis de sentimientos empiezan a trabajar identificando el nivel de análisis o granularidad del texto que tienen a su disposición (Liu, 2012).

Nivel de documento

Se aboca a la clasificación de opiniones en documentos completos tales como revistas o críticas a productos. La polaridad resultante puede ser positiva o negativa.

Nivel de oración

Estudia la clasificación a nivel de oraciones, las cuales pueden calificarse como positivas, negativas o neutrales. Este nivel está relacionado con el área de la clasificación subjetiva, que se encarga de clasificar las oraciones en dos grupos, objetivas y subjetivas. Ambos tipos de oraciones pueden expresar o no opinión, pero suele decirse que las oraciones objetivas son de polaridad neutra mientras que las subjetivas son positivas ó negativas.

Nivel de entidad y aspecto

Este nivel indica que las opiniones consisten de objetivo, aspectos del objetivo y sentimiento hacia cierto aspecto. Cada opinión debe tener un objetivo ya que de lo contrario tiene poco valor. A diferencia de los niveles de documento y oración, el nivel de entidad y aspecto se preocupa más del objeto hacia el cual la crítica es dirigida.

2.1.3. Características

La clasificación de sentimientos es la ingeniería de un grupo de características efectivas (Liu, 2012). La palabra **característica** se refiere a un atributo de los datos, una columna si estos están organizados en tablas. Las cuales normalmente son de tipo numérico (entero o real) con los cuales se puede alimentar un modelo matemático.

Se suele utilizar muestras de opiniones etiquetadas como datos de entrenamiento para el modelo, es por eso que el rumbo que normalmente toma una solución de clasificación de sentimientos es **aprendizaje supervisado**.

Desde el punto de vista del aprendizaje supervisado, el análisis de sentimientos es un problema de clasificación de textos. Los primeros intentos apostaron por utilizar características sencillas, como por ejemplo **unigramas**[1].

Debido a que los modelos matemáticos funcionan con números, las áreas de procesamiento de lenguaje natural deben transformar sus datos iniciales en una representación matemática. Áreas como el reconocimiento de voz han desarrollado técnicas para convertir frecuencias de sonido en características adecuadas, para el caso de análisis de sentimientos donde los datos crudos son textos, también existen técnicas interesantes.

Las características en clasificación de sentimientos suelen ser vectores de palabras que tienen el tamaño de una colección llamada **vocabulario**[2], donde cada elemento del vector está asociada a un término del vocabulario, el significado de este elemento depende de la estrategia a usar. Algunas de las características más usados son las siguientes:

Términos y sus frecuencias

Se calcula la frecuencia para cada ocurrencia del término dentro del texto, donde un término puede referirse a una sola palabra (unigramas) o conjuntos de palabras (n-gramas).

Una variación de este enfoque es *TFIDF* que significa, frecuencia invertida de términos y sus frecuencias en el documento. Donde primero se crea un vector con la frecuencia de las palabras de un documento y luego se calcula el inverso.

Esto se basa en la suposición de que palabras menos frecuentes, como por ejemplo, tuberculosis tienen un peso más significante en la clase del documento. Debido a que una palabra como tuberculosis tiene menos oportunidades de aparecer en un texto normal, es

[1] Un vector formado por la cantidad de ocurrencias de ciertas palabras en un texto.

[2] Palabras preseleccionadas que conforman el espacio de palabras conocidas, cualquier palabra fuera de este conjunto es ignorada.

más probable que se trate de un documento médico si es que aparece.

Partes de discurso

Permite etiquetar las palabras según su función gramatical dentro de la oración, esto logra que palabras que poseen un significado como verbos sean diferentes de otras como adjetivos. Así por ejemplo pueden manejarse adjetivos como características más importantes, ya que se sabe suelen expresar sentimientos.

Palabras y frases de sentimiento

Palabras como bueno, maravilloso, sorprendente suelen tener una valoración positiva, y otras como malo, pobre, terrible, tienen en cambio una valoración negativa. Este tipo de términos son muy importantes para la clasificación de sentimientos y puede ser interesante tratarlos con mayor relevancia.

2.2. APRENDIZAJE DE LA MÁQUINA

Mejor conocido como *Machine learning* es una subárea de la Inteligencia Artificial que devuelve resultados, logrando que la computadora aprenda cómo resolver el problema. Mitchel define *Machine learning* como "*Un programa de computadora que aprende de experiencia E con respecto a alguna clase de tareas T resultando en P, si el resultado sobre las tareas T, al igual que P, mejora con experiencia E*" (Mitchell, 1997, p. 2).

En la práctica ha probado ser de gran valor en aplicaciones de diversos dominios, tales como:

- **Minería de datos**, se infiere información resumida y valiosa a partir de grandes cantidades de datos.
- **Dominios de condiciones cambiantes**, donde el programa debe adaptarse a procesos dinámicos.
- **Dominios de poco entendimiento**, en los que no se cuenta con el conocimiento necesario para programar la solución de una tarea compleja (Por ejemplo, reconocimiento de rostros).

2.2.1. La tarea, la experiencia y la medida

El término **Tarea** (T) se refiere a la manera en la que el sistema procesa la entrada, y no así a las actividades o diferentes algoritmos aprendizaje (Goodfellow, Bengio, y Courville, 2016). Por ejemplo la tarea de Clasificación, recibe como entrada un vector de números reales y retorna un valor que indica la pertenencia a una clase entre un conjunto predefinido de ellas. Otras tareas populares son Regresión, Agrupación, Transcripción, Traducción, Detección de anomalías, etc.

La **Experiencia** (E) la componen un conjunto de ejemplos o muestras que son utilizados como entrada en un algoritmo de aprendizaje. Según el tipo de experiencia que el algoritmo está permitido a recibir estos se pueden dividir en dos categorías: *a*) Aprendizaje Supervisado donde los ejemplos están etiquetados o acompañados con su resultado esperado, utilizado en tareas como Regresión y Clasificación. *b*) Aprendizaje no Supervisado que abarca las muestras no etiquetadas, común en actividades de Agrupación (Deshpande, 2018).

Por último para determinar si un sistema realiza su tarea correctamente es necesario una **Medida** (P) que indique su correctitud. En tareas como Regresión y Clasificación existe el concepto de Acierto que es usado como medida.

2.2.2. Cómo abordar un problema

Un solución de *Machine learning* está bien definida si primero tiene clara la tarea, esto quiere decir, el problema se aborda con una técnica por ejemplo Regresión. Posteriormente se debe notar que es necesario una fuente de experiencia confiable ya que el sistema rendirá tan bien como la calidad de los ejemplos de entrenamiento.

Finalmente, se necesita una forma de medir el éxito de la solución, si los resultados no son los esperados puede ser necesario una reformulación. La medida ayuda a saber si las decisiones que se tomaron mejoraron los resultados (Mitchell, 1997).

2.2.3. La función objetivo y función de costo

Se puede ver a *Machine learning* como una forma aplicada de estadística. La labor es estimar a través de una muestra poblacional una función que pueda predecir o encontrar un patrón en dicha muestra. Tal función se conoce con el nombre de **hipótesis** o **función objetivo** $h(x)$ y es la finalidad de los métodos de *Machine learning* encontrar una función objetivo que mejor generalice un conjunto de datos.

El significado que se le da a una función objetivo depende del contexto del problema que se está resolviendo. Por ejemplo, predecir el precio de una casa para comprarla. La función retornará un valor que indique el precio en bolivianos de una casa hipotética que está definida por algunos factores. Estos factores pueden ser la ubicación, número de dormitorios, tamaño de la propiedad, los vecinos, etc. El precio de la casa es el resultado de la función objetivo, y los factores que intervienen en la decisión son la entrada.

Al estar *Machine learning* muy ligado a las matemáticas, las funciones objetivo retornan y reciben valores de tipo numérico. Si los datos iniciales no son de tipo numérico, ellos pasan primero por una etapa de procesamiento, como ya se mencionó en el caso para tareas de procesamiento de lenguaje natural en la Sección 2.1.3. Los valores de entrada de la función objetivo también se conocerán como **características** ya que es el mismo concepto.

En la siguiente función objetivo de ejemplo $h(x_0, x_1, x_2) = x_0 + x_1\theta_1 + x_2\theta_2$, se puede observar que recibe tres entradas que son evaluadas junto con unos coeficientes θ_i. Estos coeficientes son los valores que determinan la forma de la función y se conocen como **parámetros**.

Durante este capítulo se ha estado hablando sobre entrenamiento y aprendizaje pero ¿Cómo es que esto se logra en la práctica? La respuesta reside en los llamados parámetros de la función objetivos. Mediante métodos numéricos de optimización de funciones, es posible encontrar los parámetros ideales que generalicen mejor un conjunto de datos. Se puede notar que esto no está garantizado y una de las razones es la definición de la función objetivo. En el ejemplo del párrafo anterior se puede ver que $h(x)$ sólo está definida por 3 coeficientes θ, y eso es suficiente para problemas que puedan resolverse con funciones de baja dimensionalidad. Existen casos más complejos que poseen más de 3 características, ellas necesitan de una función objetivo más compleja. Saber qué función generalizará bien un conjunto de datos no es tarea fácil y es el pan de cada día en *Machine learning*.

En la siguiente sección se hablará sobre la optimización de funciones y cómo se aplica para casos de uso de *Machine learning*. Pero antes se introducirá el concepto de la función de costo, importante para hallar la función objetivo.

La **función de costo** $J(\theta)$ o también llamada función de pérdida, es la función que indica el error de la función objetivo. Representa la penalidad que obtiene la función objetivo al ser comparada con los resultados esperados. Es decir, calcula un valor que indica cuán acertado está evaluando la función objetivo con respecto al resultado esperado, dada cierta entrada.

2.2.4. Optimización

La optimización es el proceso numérico de minimizar o maximizar una función $f(x)$ por medio de alteración de x, sin embargo la minimización es lo más común. En cálculo la minimización está estrechamente ligada al concepto de la derivada.

Sea $y = f(x)$ donde x y y son valores reales, se define a la derivada de la función f en función de x como la pendiente de f en el punto x, y se denota como $f'(x)$ o $\frac{dy}{dx}$. Es decir, la derivada indica cómo escalar un cambio pequeño en la entrada con tal de obtener un cambio correspondiente en la salida. La derivada es buena para minimizar ya que indica cómo modificar la entrada x para hacer una pequeña mejora en y (Goodfellow y cols., 2016).

En *Machine learning* el objetivo es minimizar la función de costo, hacer que el error de la función objetivo sea el menor posible y así generalizar un conjunto de datos. Es frecuente mencionar entrenamiento en lugar de optimización ya que este término proporciona una mejor intuición. En este documento no se manejan diferencias entre ambos términos.

Existen diversos métodos de optimización pero el más usado en *Machine learning* es el llamado **descenso de gradiente**. Este algoritmo obtiene su nombre en honor al resultado de la derivada sobre una función multidimensional (el cual se denomina gradiente) y al hecho de que la derivada indica la dirección hacia donde está el mínimo.

El algoritmo de descenso de gradiente es un proceso iterativo que consiste en 3 etapas, evaluar $h(x)$, calcular el costo $J(\theta)$ y actualizar los parámetros de la función objetivo (denotados con θ). Este procedimiento se repite N veces, utilizando todos los datos de entrenamiento en cada iteración. Por utilizar el conjunto de datos completo en cada iteración también recibe el nombre de **descenso de gradiente por lotes**.

El algoritmo 1 muestra el método del descenso de gradiente, donde se puede ver que este tiene tres argumentos de entrada. El primero se nombra tasa de aprendizaje (α) e indica el tamaño del paso que se realiza en cada iteración rumbo hacia el mínimo. Si la tasa de aprendizaje es muy pequeña, el método tardará mucho en converger, y si es muy grande puede no hacerlo nunca. El segundo argumento es el número de iteraciones (N), el cual representa el número máximo de iteraciones que se realizarán. Debido a que el descenso de gradiente es un algoritmo iterativo puede que nunca se llegue a converger, es por ello que se necesita un seguro para no terminar en un ciclo infinito. El último argumento es el conjunto de entrenamiento que tiene dos partes, uno se denomina (X) y representa a los datos de entrada de la función objetivo, la otra parte (Y) contiene los resultados esperados de cada ejemplo en (X).

Algoritmo 1 Descenso de gradiente

Input *Tasa de aprendizaje* (α)
Input *Número de iteraciones* (N)
Input *Conjunto de entrenamiento* (X, Y)
Output *Parámetros entrenados* (θ)

1: $\theta \leftarrow 0$
2: **for** N iteraciones **do**
3: $\quad h \leftarrow h(\theta, X)$
4: $\quad \hat{g} \leftarrow J'(\theta)$
5: $\quad \theta \leftarrow \theta - \alpha\, \hat{g}$
6: **end for**
7: **returns** θ

Al ser el descenso de gradiente un proceso iterativo que se realiza N veces, se espera que las iteraciones sean suficientes para que la función de costo converja[3] en su mínimo. Dependiendo del método de *Machine learning* que se esté empleando la función objetivo y de costo elegidas varían, modificando a su vez el algoritmo en mayor o menor medida. Algunas funciones de costo tienen una forma convexa[4], eso quiere decir que solo existe un mínimo global[5] por lo que la convergencia está casi garantizada. Otros modelos más complejos no necesitan converger para obtener buenos resultados, estos pueden sentirse satisfechos con al menos alcanzar un mínimo local[6].

2.2.5. Hiperparámetros

Los hiperparámetros son los valores, usualmente constantes, que determinan el diseño o entrenamiento de un modelo de *Machine learning*. A diferencia de los parámetros, que se aprenden durante el entrenamiento, son valores ingresados que determinan cómo se van a aprender tales parámetros (por lo que la comunidad ha determinado utilizar el prefijo "hiper").

El problema de encontrar el conjunto correcto de hiperparámetros es una preocupación muy recurrente en *Machine learning*. Al ser insertados manualmente introducen incertidumbre ya que no se conoce con certeza los valores ideales, pero con el tiempo se han desarrollado métodos que ayudan a encontrar valores lo suficientemente buenos. En este documento no se indagará mucho más sobre este asunto, fuentes más especializadas como (Christopher, 2006) tienen más información.

[3] Aproximarse a un límite.
[4] Que tiene forma de cono.
[5] Punto de una función que se encuentra en la posición más baja de esta, representa el límite inferior.
[6] Mínimo local es un mínimo pero que no es el más inferior.

La tasa de aprendizaje y el número de iteraciones, valores antes mencionados en el algoritmo de descenso de gradiente, son los primeros hiperparámetros presentados en este documento. A medida que se desarrolle la hipótesis se explicarán más hiperparámetros.

2.3. LAS REDES NEURONALES

Las redes neuronales son un método que generaliza datos componiendose de varias funciones no lineales al mismo tiempo. Se originaron en la búsqueda de una representación de la información y procesos que se llevan a cabo en los sistemas nerviosos biológicos. De este movimiento surgió un grupo que apuntaba a desarrollar algoritmos efectivos de aprendizaje, logrando avances sin necesidad de copiar al pie de la letra el funcionamiento del cerebro (Mitchell, 1997).

Son usadas para problemas de clasificación y alternativa a métodos clásicos como *SVM*[7] ya que ambos tienen una salida probabilística y rendimiento similar. A diferencia de las redes neuronales, *SVM* es un clasificador de margen grande[8] y además tiene una función de costo convexa (Christopher, 2006).

Es apropiado utilizar redes neuronales en problemas donde los datos de entrenamiento tienen errores, ya que el algoritmo de propagación del error ***backpropagation***, es tolerante a muestras ruidosas. Escenarios donde una evaluación rápida de la entrada es necesaria también son propicios, ya que predicen más rápido que los clasificadores *SVM*, pero a diferencia de ellos toman mucho más tiempo en la etapa de entrenamiento (Mitchell, 1997).

Una preocupación recurrente de las redes neuronales es el **sobre entrenamiento**[9], por lo cual métodos de **validación cruzada** o **regularización** pueden ser usados para estimar cuándo detener el entrenamiento, determinar los hiperparámetros adecuados o evitar aprender funciones muy complejas, ya que tienden a sobre entrenarse.

2.3.1. El perceptrón

Cuando se explica el funcionamiento interno de una red neuronal siempre es más fácil empezar por la unidad más sencilla y fundamental de esta, el perceptrón.

[7] Acrónimo para Support Vector Machine.

[8] El límite de decisión de la función objetivo considera cierto margen entre los ejemplos situados en el borde de dos clases.

[9] Generalizar excelentemente el conjunto de entrenamiento tanto que el rendimiento al evaluar sobre datos nuevos es muy pobre.

Un **perceptrón** es una unidad lógica que también tiene el nombre popular de neurona artificial. En *Machine learning* esta es una técnica que es mejor conocida como **regresión logística**. En este documento se utilizaŕa el término perceptrón o neurona para referirse a este clasificador.

Se puede utilizar una neurona por si sola para tareas de clasificación binaria, aunque suele necesitar la ayuda de un **umbral** para realizar las predicciones. Una neurona devuelve resultados que se encuentran en un rango de 0 a 1, así que con la ayuda de un umbral se determina el límite entre clases.

Es una unidad fundamental para conformar redes neuronales. Agrupaciones de neuronas permiten aprender funciones mucho más complejas, y también tener mayor cantidad de salidas. Las redes neuronales pueden realizar predicciones de más de dos clases gracias a la composición de varios perceptrones en un solo clasificador.

Entrando en los detalles del perceptrón, la función objetivo de esta se divide en dos partes, la primera se llama **activación lineal** y se calcula como sigue,

$$z = w^T x + b$$

donde w y x son vectores de unidimensionales y la operación entre ambos es conocida como producto punto en álgebra lineal. Los parámetros del perceptrón son w y b, donde el vector w debe resultar familiar ya que representa a los coeficientes de la función. El parámetro b es el conocido **bias** el cual no se multiplica por ninguna entrada y se suma aparte (tener el bias como un término aparte es una convención popular en *Machine learning*).

La segunda parte de un perceptrón se llama **activación no lineal** y consiste en pasar el resultado de la activación lineal calculada anteriormente por medio de una función no lineal g.

$$\hat{y} = g(z)$$

Esta función se aplica a cada elemento resultado de la activación lineal y es utilizada para forzar, por decirlo de alguna manera, una no linealidad de la función objetivo.

Para clasificadores sencillos, por ejemplo una neurona, la ***sigmoide***, es la función que normalmente se usa en la etapa de activación no lineal,

$$sigmoide(z) = \frac{1}{1+e^{-z}}$$

la cual tiene salida probabilística con rango entre 0 y 1. Más adelante se verán otras

funciones de activación, más apropiadas para redes neuronales.

En la Figura 1 se resume el funcionamiento interno de una neurona. La activación lineal se grafica como la sumatoria de la multiplicación de la entrada por los pesos del perceptrón. Se puede ver que el resultado de la etapa anterior se utiliza como entrada para la función de activación, que en este ejemplo utiliza un umbral para predecir el resultado.

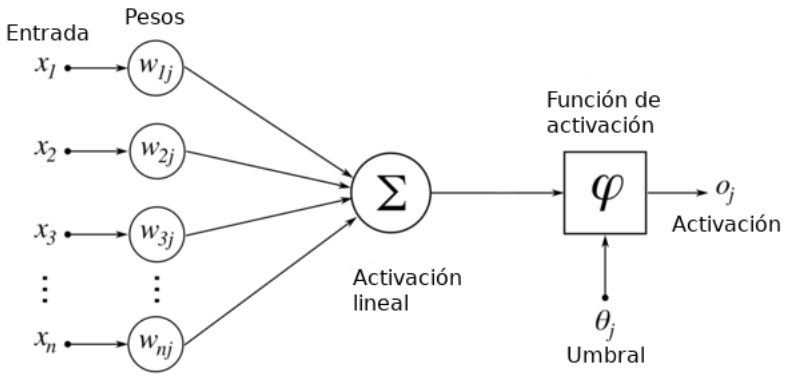

Figura 1: Ejemplo de un perceptrón (Adaptado de Tom Mitchell 1997).

Este clasificador puede ser entrenado con el algoritmo de descenso de gradiente. El gradiente como se explicó en una sección anterior, se calcula derivando la función de costo. Dependiendo de la función de costo elegida, la derivada puede ser más o menos difícil de calcular, pero esos son detalles que se cubrirán más adelante.

2.3.2. La composición de una red neuronal

Las redes neuronales, han tenido muchos nombres a lo largo del tiempo, uno de ellos y de los más viejos es perceptrón multicapa, otros son red neuronal artificial, completamente conectada, superficial, etc. En este documento el término redes neuronales se referirá a ellas en general y se nombrarán tipos específicos de redes cuando sea el caso.

Una red neuronal como su propio nombre lo indica está conformado por neuronas, más precisamente conjuntos de neuronas agrupadas e interconectadas entre sí por niveles o **capas**. La capa de entrada es la primera capa de una red neuronal y la última es la capa de salida, cualquier capa intermedia entre la capa de entrada y la de salida es llamada **capa escondida**, ya que es transparente para quién introduce los datos.

Se suele conocer a estas máquinas como clasificadores de caja negra, ya que aprenden funciones tan complejas que no se sabe cómo interpretarlas. A pesar de ser una caja negra, se conoce la composición interior de una red neuronal. A la forma de una red neuronal se

conoce como **arquitectura** o diseño de la red. La arquitectura a elegir es una decisión en la que se debe tomar en cuenta: El problema a resolver, la cantidad de datos, capacidad de procesamiento, y precisión esperada. Una sola arquitectura no funciona para todos los problemas y es por ello que diversos tipos de redes neuronales han surgido (por ejemplo las redes neuronales convolucionales), pero una arquitectura que funciona para cierto problema puede funcionar para uno similar.

En este documento se diferenciará tipo de red neuronal de arquitectura. Un tipo se refiere a una técnica basada en una red neuronal clásica que añade alguna operación o realiza las evaluaciones de forma diferente. Mientras que una arquitectura indica la forma de un tipo de red neuronal, por lo que un tipo de red neuronal puede ser implementado con distintas arquitecturas.

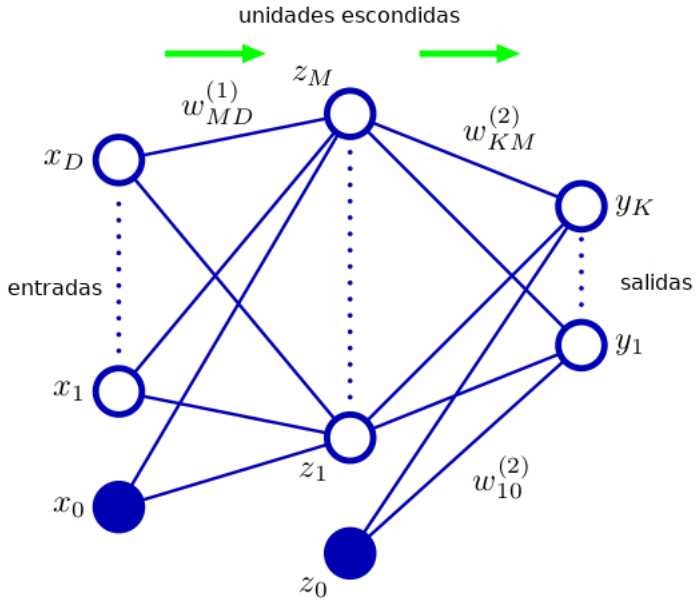

Figura 2: Ejemplo de la composición interna de una red neuronal (Adaptado de Bishop M. 2016).

Existen varios tipos de redes neuronales, la más popular se entiende mejor al escuchar el nombre de **red profunda de propagación hacia adelante**, la cual se llama así debido a la conexión entre sus capas intermedias y la peculiaridad de estar encadenadas en una dirección. Quiere decir, que evalúan propagando sus resultados de un nivel al siguiente sin tener ningún tipo de retroalimentación de capas anteriores (Goodfellow y cols., 2016). Las redes que permiten tener retroalimentación de capas anteriores se llaman **redes neuronales recurrentes**. Uno de los objetos de estudio de esta investigación y tipo de red popular, las **redes neuronales convolucionales** se explicarán con más detalle en secciones posteriores.

En la Figura 2 se muestra un ejemplo de la composición interna de una red neuronal, se puede ver que está dividida por columnas (capas) y estas a su vez contienen otras unidades lógicas (neuronas) interconectadas entre sí. La cantidad de capas que una red neuronal tiene se denomina **profundidad** (L), y es por la traducción de este término en inglés que *Deep learning* tiene su nombre. Una red neuronal con una profundidad pequeña se conoce como **superficial**, en contraparte si tiene muchas capas es **profunda**. Entre capas se encuentra la función de activación (g) que puede ser diferente entre capas pero la misma para todas las neuronas dentro una.

Cada neurona tiene sus propios parámetros los cuales en conjunto se conocen también como **pesos** (W), así se suelen nombrar a los parámetros de una red neuronal. A diferencia del perceptrón cuyos parámetros son vectores, una red neuronal conformada por varias neuronas tiene matrices como parámetros, indicando la contención de varios perceptrones. Por otro lado el bias (b) también aumenta su dimensión, de real a vector unidimensional. Cada capa de una red neuronal tiene sus propios parámetros (pesos y bias) que no necesariamente comparten dimensiones entre capas, depende de la arquitectura de la red.

2.3.3. Terminología

Los parámetros principales de una red neuronal son la matriz de pesos simbolizados con W y el bias representado con b. Como se mencionó antes el bias es un vector cuando se trata de redes neuronales.

Durante la etapa de activación lineal la matriz intermedia que se genera se denomina Z, pero luego de pasar por la activación no lineal se llama A. La función de activación se denominará g, recordar que cada capa puede tener una función diferente.

Para identificar los términos entre las diferentes capas de una red neuronal se cuenta con el superíndice l, la profundidad e índice de la última capa será L. Estos índices se aplican a parámetros, activaciones y funciones de activación. Por otro lado el subíndice i denota un elemento específico de una matriz, no será muy frecuente pero se puede observar su uso en la derivada parcial del costo.

Como se puede notar las letras mayúsculas se usan para denotar matrices haciendo alusión a su dimensión más elevada. Las letras minúsculas se reservan para vectores o números reales. Hasta ahora se ha estado mencionando los nombres matrices o vectores para indicar estas estructuras multidimensionales, pero más adelante también se usará el nombre de **tensor** el

cual es término más actual para referirse a un vector multidimensional[10].

2.3.4. Entrenamiento

Por simplicidad esta explicación tomará como base el algoritmo de descenso de gradiente (Ver Algoritmo 1) y una red neuronal superficial, pero el mismo principio se aplica a diferentes tipos de redes neuronales.

El entrenamiento de una red consta de tres pasos, el primero se llama **propagación hacia adelante**. Consiste en propagar las activaciones de la primera capa, hacia la siguiente como si se tratara de la entrada. Y así hasta la última capa obteniendo la predicción como resultado. Se puede ver a la propagación hacia adelante como la evaluación de la función objetivo $h(x)$. Nótese que en esta sección se ha dejado de mencionar a la función objetivo, y es que la red neuronal representa a la función objetivo.

$$Z^l = W^{lT} . A^{l-1} + b^l$$

$$A^0 = X$$

$$A^l = g^l(Z^l)$$

$$\ldots$$

$$\hat{y} = A^L$$

El segundo paso es la **propagación hacia atrás** donde al contrario del paso anterior se comienza desde el final. Los detalles de este procedimiento está fuera del alcance de este documento pero el algoritmo que se utiliza para este propósito se conoce como ***backpropagation***. En resumen, la propagación hacia atrás no es más que el cálculo del gradiente, resolviendo la derivada de la función de costo $J(\theta)$.

Finalmente el tercer paso consiste en **actualizar los pesos**, con la regla de actualización de parámetros $\theta \leftarrow \theta - \alpha\ \hat{g}$.

$$W^l = W^l - \alpha\ \frac{\partial J(\theta)}{\partial W_i^l}$$

$$b^l = b^l - \alpha\ \frac{\partial J(\theta)}{\partial b_i^l}$$

[10]Que tiene n dimensiones. Ejemplo de un arreglo de 2 dimensiones, una matriz.

Deben actualizarse todos los pesos de todas las capas con el gradiente correspondiente. Ya que se tienen dos parámetros que optimizar ambos son actualizados, todo esto dentro un bloque iterativo como indica el algoritmo de descenso de gradiente.

2.4. APRENDIZAJE EN PROFUNDIDAD

También conocido como *Deep learning*, es el paradigma que estudia ciertas estructuras lógicas que interconectadas representan el funcionamiento del cerebro. Cada una de estas estructuras personifica conceptos que pueden ser sencillos por si solos, pero unidas entre sí componen máquinas complejas y poderosas (Goodfellow y cols., 2016). Puede entenderse como una implementación de *Deep learning* a una red neuronal con más de una capa oculta.

Su auge es reciente, ya que con el pasar de los años las computadoras se han vuelto más poderosas y capaces de mantener redes más complejas y profundas. La neurociencia también ha tenido gran relevancia en la investigación de redes neuronales más profundas, aportando con nuevos conceptos y arquitecturas que se basan en cerebros biológicos reales.

Las técnicas de *Deep learning* han probado ser buenas en campos como reconocimiento de voz, procesamiento de lenguaje natural, robótica, etc. Pero el área donde mayor éxito ha tenido es visión por computadora gracias a su capacidad de extraer información en distintos niveles de detalle.

Desde 2012 *Deep learning* ha ido creando una gran reputación en la solución de problemas de visión por computadora debido a una arquitectura de red neuronal convolucional, propuesta por Krizhevsky, Sutskever, y Hinton (2012) que ganó con gran diferencia la competencia de ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*). Desde entonces una red de convolución siempre gana estas competencias. Las redes neuronales convolucionales se convirtieron en el estado del arte para la tarea de reconocimiento de imágenes.

Pero esto no quiere decir que las técnicas de *Deep learning* no puedan ser usadas en otros dominios. También existe investigación del uso de redes neuronales convolucionales para procesamiento de lenguaje natural como Collobert y cols. (2011). La tendencia popular de indicar que las redes neuronales convolucionales solo funcionan para reconocimiento de imágenes es debido a que existe una interpretación muy conveniente para este caso: A medida que una imagen es procesada por capas de convolución, la primera capa reconoce líneas, luego bordes, figuras, y así hasta reconocer objetos complejos. En cambio para procesamiento de lenguaje natural, no existe una interpretación tan elegante, pero se quiere

creer que aprenden parámetros que reconocen características similares a n-gramas.

2.5. REDES NEURONALES CONVOLUCIONALES

Cuando se trata de realizar predicciones en casos donde la salida no cambia, debido a una o más transformaciones de la entrada, las redes neuronales clásicas tienen deficiencias ya que su representación no tiene en cuenta el manejo de esta restricción. Por ejemplo, en reconocimiento de rostros una red normal puede manejar sin mucho problema ejemplos con caras de frente, pero si se encontrarán rotadas, trasladadas o un poco deformadas, ya no es tan fácil. Con suficiente tiempo y grandes cantidades de datos, tal que agrupen todas las transformaciones, se lograría que la red clásica aprenda a manejar las variantes. La mejor opción en estos casos es usar otro tipo de red, más especializada y eficiente las llamadas redes neuronales convolucionales (Christopher, 2006).

Las redes convolucionales, son solo otro tipo de red neuronal que en lugar de utilizar la multiplicación de matrices utiliza la **convolución** en alguna de sus capas (Goodfellow y cols., 2016). Pero en realidad otros conceptos además de la convolución suelen ser añadidos a estas redes. Una definición más general indica que son una solución a la **invariancia**[11], logrando crear propiedades que tratan esta cualidad dentro de la estructura de la red (Christopher, 2006).

Son redes que se utilizan ampliamente en el área de la visión por computadora (Christopher, 2006), aunque pueden ser aplicadas a cualquier tipo de entrada que tengan una topología parecida a un cuadrícula, es decir arreglos n-dimensionales o tensores. Un ejemplo de arreglos de dos y tres dimensiones son las imágenes con el correspondiente número de canales de color. Una representación de una dimensión pueden ser las series de tiempo (Goodfellow y cols., 2016).

2.5.1. Ventajas

Las redes neuronales convolucionales tienen algunas características ventajosas a mencionar. Primero las **interacciones esparcidas**, los parámetros de una capa solo interaccionan con un pequeño conjunto de parámetros de la siguiente capa, permitiendo que se activen solo ciertas unidades ignorando completamente el resto de ellas. A diferencia de una red neuronal clásica donde todas las unidades entre dos capas contiguas interactúan (red

[11] La salida de una función no debería cambiar si la entrada sufre una o más transformaciones.

completamente conectada). Esto tiene un gran impacto en el rendimiento de la red.

Otra ventaja tiene que ver con el hecho de que los **parámetros se reusan** en diferentes secciones de la entrada, lo que significa ahorro de procesamiento en la etapa de entrenamiento y menos almacenamiento requerido ya que la cantidad de parámetros es mucho más pequeña que la entrada.

Una última característica ventajosa de la convolución, es que produce **representaciones equivariantes**. Si la entrada cambia la salida cambia de la misma forma. Aunque esto solo se limita a transformaciones de traslación, operaciones como escalado y rotación son manejados con otro concepto llamado ***pooling***.

2.5.2. Convolución

Cuando se implementa una red convolucional la diferencia fundamental con las redes clásicas es que la operación de multiplicación entre matrices, es reemplazada por la operación de convolución. El operador $(*)$ simboliza la convolución como se puede ver en el siguiente ejemplo.

$$Z^l = A^{l-1} * W + b^l$$

La matriz W se llamada ***kernel*** o **filtro**, normalmente tiene tamaño impar y forma cuadrada, por ejemplo 3×3 o mayores. En el ejemplo A es la matriz de activación de la capa anterior, pero cuando se trata de la operación de convolución se suele llamar también **mapa de características**. Se puede ver también que el bias se suma de forma usual, pero se debe tener en cuenta que la dimensión de la matriz resultado de la convolución es reducida en comparación a la entrada, y se sabe que el bias debe tener la misma altura que la salida.

La operación consiste en que el filtro se desplaza a través del mapa de características. Para cada movimiento se calcula el producto, entre cada elemento del filtro y el elemento de la entrada con la que se sobrepone. Luego los resultados se suman y la salida es colocada en la posición correspondiente (Dumoulin y Visin, 2016).

Debido a la naturaleza de la operación, los valores en los bordes del mapa de características son usados una sola vez en la convolución, esto en general no es un problema pero dependiendo de la aplicación se puede querer aplicarlos mejor. Se puede utilizar una estrategia de relleno que aumenta la dimensión del mapa de características y llena los bordes con ceros, aprovechando más los valores antes orillados.

El relleno también se puede utilizar para evitar reducir el tamaño de la matriz resultante

si es que se quiere mantener la dimensión del mapa de características entre capas de convolución. La siguiente fórmula resume las dimensiones de la salida.

$$\left(\left\lfloor \frac{n_H - f + 2p}{s} + 1 \right\rfloor, \left\lfloor \frac{n_W - f + 2p}{s} + 1 \right\rfloor\right)$$

Donde f es el tamaño del filtro, y los valores n_H, n_W son el ancho y alto del mapa de características respectivamente. Los símbolos $\lfloor\ \rfloor$ representan la operación matemática ***floor***[12].

La variable p representa el relleno antes mencionado. Indica cuánto crecerá el mapa de características en ancho y alto, el valor frecuente de p es 2, uno a la izquierda y uno a la derecha, asimismo para arriba y abajo. Con un valor de relleno $p = 2$ se logra que la salida conserve el tamaño original del mapa de características.

El valor de la variable s indica el paso, es decir, cuánto se moverá la ventana de convolución (el filtro a través del mapa de características) horizontal y verticalmente (Dumoulin y Visin, 2016).

La operación de convolución antes descrita se conoce como convolución en dos dimensiones ya que la ventana deslizante (el filtro) es una matriz que se desplaza en las dos dimensiones del mapa de características. Una convolución de tres dimensiones es posible cuando se tienen varios mapas de características apiladas donde la cantidad de ellas se registra como **canales** (por ejemplo una imágen con tres canales de color). En ese caso una convolución de tres dimensiones tiene ventanas en la forma de cubos, las cuales se desplazan a lo largo del ancho, alto y canal (Dumoulin y Visin, 2016).

En la literatura referente a *Deep learning* los hiperparámetros paso y relleno se conocen normalmente con los nombres en inglés *stride* y *padding* respectivamente, de la primera letra de estos términos vienen los símbolos que representan a estos valores (s y p). Con tal de mantener constancia con las fuentes en inglés se mantendrá la simbología estándar para estas variables, pero se traducirá su nombre.

2.5.3. *Pooling*

Al momento de diseñar una red neuronal convolucional, se puede considerar que la capa de convolución contiene tres etapas dentro. La primera etapa es la convolución, durante la cual se realiza la operación mencionada para producir un conjunto de activaciones lineales.

[12]Operación que aproxima un número real hacia el entero menor más próximo.

La segunda etapa se suele denominar detector, donde cada activación lineal es corrida ante una función no lineal. Por último, en la etapa tres se utiliza una función de *pooling* para modificar aún más la salida (Goodfellow y cols., 2016).

La función de *pooling* reemplaza a la función de salida de la red con un resumen estadístico de una sección de los resultados, esta puede ser por ejemplo, ***max-pooling*** que retorna el valor máximo de un área rectangular, u otras funciones como ***promedio***, o ***norma*** $\boldsymbol{L}^2$ también son usadas pero con menos frecuencia.

El *pooling* ayuda a que la representación sea invariante a pequeñas transformaciones de la entrada, esta es una propiedad útil dentro una red neuronal, en especial para los casos en los que importa saber más si un objeto está presente o no, que saber dónde está exactamente. Se mencionó anteriormente que la convolución no solventa por sí misma el escalado y la rotación, ya que la etapa de *pooling* solventa estas preocupaciones.

Esta operación también modifica el tamaño de la matriz resultante. La fórmula antes mencionada para calcular la dimensión de la salida de la convolución se puede usar también para *pooling*. La única diferencia es que *pooling* no suele usar relleno, así que se puede suponer que $p = 0$.

CAPÍTULO 3 PROCESAMIENTO DE DATOS

Debido a la facilidad de las empresas o instituciones de obtener grandes cantidades de datos en poco tiempo, los sistemas inteligentes han tenido un gran auge comercial y académico. Esta tendencia ha crecido por un lado porque ayudan a realizar estudios de mercado que mejoran la experiencia de usuario, otro porque investigaciones novedosas con modelos más complejos son posibles. En especial *Deep learning* tuvo un fuerte empujón aquí, ya que los diseños de redes neuronales que se proponen empiezan a funcionar de verdad con conjuntos de datos que van desde miles a millones de ejemplos. Hace 10 años se entrenaban pequeños reconocedores con poco más de 100 datos pero hoy en día eso no es suficiente.

Desde que una de las metas de esta investigación es evaluar el acierto de un modelo[1] sobre un conjunto de datos pertenecientes al turismo en Bolivia, se requiere información que cumpla tales propiedades. Por desgracia no existe una publicación de datos de tal tipo, ese es el motivo por el que fue necesario conformar un conjunto de datos como parte de esta investigación.

Se logró crear un conjunto de datos pequeño en comparación a los usados en sistemas comerciales, pero aún así, tiene valor académico y es suficiente para entrenar un modelo experimental. En las próximas secciones se describirán con más detalle las propiedades, preprocesamiento y mapeo de los datos.

3.1. EL *CORPUS*

Un *corpus* es una colección de textos almacenados en formato electrónico que están seleccionados de acuerdo a ciertos criterios, su composición prioriza el mensaje que el texto quiere comunicar sobre el lenguaje (Sinclair y Wynne, 2005).

[1] Término usado para referirse a la combinación del algoritmo de entrenamiento, el diseño de una red neuronal y los datos que se usan para entrenarla.

A diferencia de las ramas lingüistas donde los detalles del lenguaje son muy importantes ya que son los objetivos de su análisis, en procesamiento de lenguaje natural se suele utilizar el término ***corpus*** para referirse simplemente a una colección de textos sin tener muy en cuenta la correctitud o incluso ortografía del texto, pero si el dominio y contexto de los datos.

Se dice que están almacenadas en formato electrónico ya que no tiene sentido que sea de otra forma. La razón de tener un *corpus* es que se puede analizar y obtener conclusiones sobre él. La mejor forma de analizar un *corpus* es con el uso de programas de computadora, especializados en análisis de textos (Evans, 2007).

Debido a la naturaleza de esta investigación el *corpus* a usar debe pertenecer a un entorno turístico de Bolivia, específicamente comentarios sobre lugares o atractivos turísticos del país. Con tal de realizar análisis de sentimientos, los datos de interés son comentarios escritos que se vierten en diversos portales como páginas de internet, foros y redes sociales.

No se entrará en más detalle sobre las fuentes u origen de los comentarios, ya que algunos de los datos son invenciones propias o traducciones de comentarios en inglés. Esto puede o no tener un efecto negativo en el desempeño del modelo pero es algo de lo que se tratará más adelante cuando se hable de la calidad de los datos.

Los comentarios que conforman el *corpus* están etiquetados, de tal forma que se puedan usar en clasificadores de aprendizaje supervisado. Cada ejemplo en el conjunto de datos puede verse como una tupla *(Comentario, Valoración)*, donde *Comentario* es un texto en español de longitud no determinada y *Valoración* es un número entero positivo entre 1 y 3. La Tabla 1 indica con claridad la relación entre una etiqueta y su valor.

Etiqueta	**Valor**
Negativo	1
Neutral	2
Positivo	3

Tabla 1: Etiquetas del *corpus*.

3.2. CRITERIOS DE SELECCIÓN

Ya se sabe qué es un *corpus*, pero aún no se ha hablado sobre las acciones a seguir durante el proceso de recolección. En esta sección se tratará de cubrir un primer paso, definir las reglas que determinan si dado un comentario es propicio para pertenecer al *corpus*.

Las reglas o restricciones que sirven como criterios de selección son diversas por lo que

solo se nombraran los que tenga más relevancia acorde a los objetivos de este trabajo. El primer criterio es el **tema**, la definición de un contexto común para todos los ejemplos a recolectar. El **dominio**, el cual se refiere al tipo de lenguaje o modo de expresión que se usa en el texto, por ejemplo un dominio académico es diferente a uno popular ya que en él se suele tener un lenguaje más refinado sin errores ortográficos ni vulgaridades. Por otro lado un dominio popular tendría menos cuidado sobre las reglas gramaticales, además usaría jergas y onomatopeyas.

Otros criterios como la **localidad** y el **idioma** representan exactamente su significado y normalmente van de la mano. A pesar de su aparente trivialidad son muy importantes en tareas de clasificación ya que el espectro que forman unos comentarios en español es muy diferente al que genera uno de opiniones en inglés. No se puede esperar que un clasificador entrenado con comentarios en inglés funcione con textos en otro idioma. Lo mismo se aplica para la localidad, la jerga o el acento hacen gran diferencia cuando se trata de entender una grabación de voz. Si para las personas a veces es muy difícil realizar la misma tarea, entonces lo es aún más para una computadora.

El **tipo** es un criterio que indica el medio de origen de los datos, ya sea un libro, revistas, videos, periódicos o la web. Por último se manejará el criterio de selección **fecha** que no tiene especial importancia en este trabajo pero mientras más recientes sean los datos mejor.

Criterio	Valor
Tema	Opiniones sobre atractivos turísticos
Dominio	Popular
Idioma	Español
Localidad	Bolivia
Tipo	Textos escritos en la web
Fecha	Más recientes de preferencia

Tabla 2: Criterios de selección.

Como se puede ver en la Tabla 2 el dominio de los comentarios elegido es “Popular”, porque se supone que estos textos no toman muy en serio las normas gramaticales, aunque no se puede asegurar nada ya que esta es una decisión totalmente subjetiva de quién escribió la opinión. Como es de esperarse los criterios de idioma y localidad son “Español” y “Bolivia” respectivamente, de acuerdo a los objetivos de esta investigación. Cabe destacar que aunque el idioma sea español y los objetivos de los comentarios sean atractivos turísticos de Bolivia, los autores no tienen que ser necesariamente del país o hablar el idioma.

Se limitó a almacenar comentarios publicados en páginas web y foros, ya que no se contaba con el tiempo ni recursos para buscar comentarios en otros medios como periódicos.

En la Tabla 2 el tipo simplemente indica “Textos escritos en la web” para referirse de manera general a los portales utilizados.

Luego de definir los criterios de selección, los siguientes pasos son encontrar, almacenar y procesar los datos. Por simplicidad, desde este punto en adelante se supondrá que ya se recolectaron los datos obedeciendo las reglas antes mencionadas, y que se almacenaron crudos en una base de datos local. El procesamiento de los datos se tocará en la Sección 3.5.

Al revisar el *corpus* conformado se encontró que la mayoría de los comentarios son textos largos y de tamaño variable. El tamaño también pudo funcionar como criterio de selección pero como se supuso es variable; no es algo de lo que se quiera preocupar en una etapa temprana del trabajo. El tamaño de las opiniones es una inquietud que se debe solucionar, se mencionará cómo se hizo durante el preprocesamiento (Sección 3.5).

Resultó difícil encontrar comentarios negativos durante la recolección, esto se pudo deber a dos factores: *a*) La calidad de los atractivos turísticos, ya que los turistas en general tienden a tener experiencias agradables sus opiniones son mayormente positivas. *b*) Se supuso que los sitios con menor cantidad de comentarios y menos calificación general tienen más críticas negativas pero no es el caso, solo son menos conocidos. De hecho los lugares más populares son los que tienen más críticas negativas pero aún así no son las suficientes como para conformar un *corpus* grande en poco tiempo.

3.3. PROPIEDADES DEL *CORPUS*

Cuando se recolectan datos, información adicional como fecha, título, autor, ciudad y otros agregan valor al *corpus* convirtiéndolo en una herramienta aún más poderosa. Esta información se conoce como metadatos y describen aún más la información que se tiene, pero se decidió no contar con ello. Se quería conservar el anonimato de los autores y mantener el *corpus* lo más sencillo posible, por lo que se evitó recolectar más de lo necesario para esta investigación. Aunque el objetivo hacia el cual la opinión es dirigida es un metadato que si se registro.

En esta sección se conocerán como propiedades, no a los recién definidos metadatos sino, a términos que ayudan con la tarea de entender la calidad del *corpus*, son conceptos basados en Evans (2007).

3.3.1. Representatividad

La representatividad en análisis de textos es un concepto difícil de definir y medir, aún así se puede entender como, un espectro que es creado por los ejemplos que conforman el *corpus*, tal que este espectro representa una muestra importante y no trivial de una población (Sinclair y Wynne, 2005). Esta propiedad es algo que debe estar en la mente del creador del *corpus*, pero es una pérdida de tiempo preocuparse demasiado por ello.

Una población se define como un conjunto de datos, que puede ser un subconjunto de otro más grande, determinado por una lista de restricciones. En secciones anteriores ya se definieron las restricciones (criterios de selección) que determinan este espacio de datos. Pero debido a la incertidumbre del tamaño de la población no es posible saber si una muestra obtenida es representativa. Se solventará este problema usando un objetivo, es decir una cantidad de datos alcanzable y que sean suficientes para los fines de esta investigación. **Se dirá que el *corpus* conformado es representativo si se alcanza el tamaño objetivo**. Al tener un objetivo, el creador del *corpus* puede mantenerse enfocado en esa meta y no preocuparse demasiado por los efectos colaterales.

En esta sección se trató el tamaño del *corpus* viéndolo como la cantidad de elementos que hay en un conjunto, es decir el interés estaba enfocado en cuántos comentarios hay. Más adelante se abordará el tamaño como la longitud de los comentarios que están en el *corpus*.

3.3.2. Balance

Durante el proceso de recolección, aparte de los criterios de selección, también se tuvo en cuenta que la cantidad de ejemplos que entran en cada una de las etiquetas antes definidas sea pareja. No se puede esperar que un clasificador de análisis de sentimientos funcione correctactamente teniendo a 90 % de los comentarios con polaridad positiva y el resto negativa. Esto se conoce como balance y es una propiedad fundamental en tareas de clasificación, pero que resulta ser costosa de conseguir.

La recolección demoró bastante porque la cantidad de comentarios negativos disponibles era muy poca y es por esto que se tuvo que recurrir a técnicas antes mencionadas, como traducciones e invenciones, para completar el *corpus*. La poca disponibilidad de ejemplos negativos también es la razón por la que el tamaño total del *corpus* es limitado.

Se dirá que el *corpus* está balanceado si, independientemente del tamaño total de este, **la cantidad de comentarios distribuidos entre las diferentes etiquetas es la misma o la diferencia entre ellos es como mínimo el 1 % del tamaño total del *corpus***.

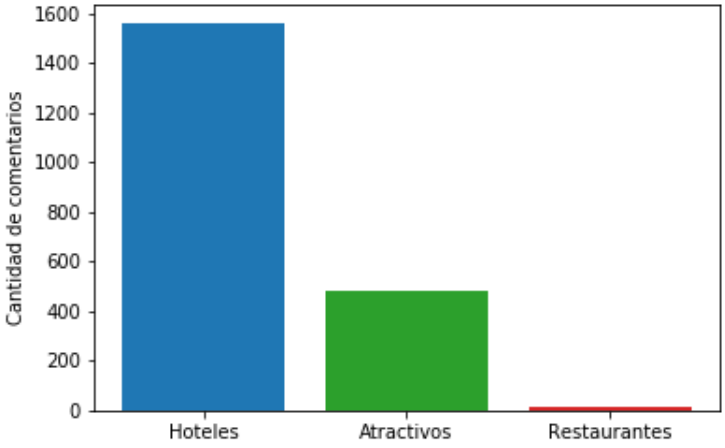

Figura 3: Distribución de objetivos de los comentarios (Elaboración propia).

Como nota aparte, también se puede hablar de balance en otras partes como el objetivo de los comentarios, es decir a quién o qué está dirigida la opinión. En la Figura 3 se puede ver una gráfica de la distribución de los objetivos de las opiniones recolectadas. Se hubiera querido lograr un balance más parejo al igual que las etiquetas pero no se tuvo en cuenta ya que no es una característica de alta prioridad, por lo que se espera que esto no tenga un efecto negativo en el desempeño del modelo.

Como se puede ver en la Figura 3 los objetivos de recolección predominante son hoteles, preferido por la abundancia de comentarios en este dominio. El segundo pico más alto indica "Atractivos" refiriéndose a lugares que son frecuentemente visitados por turistas, tales como plazas, parques, mercados, templos y en el caso de la ciudad de Cochabamba el Cristo de la Concordia.

3.3.3. Longitud

El análisis de sentimientos es una práctica muy popular entre las tareas de procesamiento de lenguaje natural y gracias a *Twitter* esta práctica se ha hecho aún más común y sencilla de lograr. Esta red social tiene políticas que hacen muy fácil la obtención de datos a través de su API[2]. Los *tweets*, como se llaman a los mensajes de *Twitter*, tienen la propiedad de tener un máximo número de caracteres, el cual es un factor determinante en análisis de texto. **Si el tamaño de los textos que se van a analizar tienen un tamaño fijo o predecible se puede asegurar que no se perderá información cuando estos sean usados en algún algoritmo**

[2]Interfaz de programación de aplicación

de *Machine learning*, este es el criterio de calidad que se desea cumplir.

En esta sección se hablará de tamaño como la longitud de un texto que es parte del *corpus*, mejor dicho la cantidad de palabras de un comentario. El *corpus* recolectado no cuenta con la ventaja de tener longitudes predecibles, es por ello que esta indeterminación debe tratarse de alguna forma. Existen diversas opciones que pueden resolver este problema, dos son: *a*) El preprocesamiento de datos. *b*) Manejarlo de alguna manera en la estructura del modelo así el tamaño de los comentarios sería trivial al no existir pérdida de datos. Más adelante en este documento se detalla el tratamiento realizado para solventar esta preocupación.

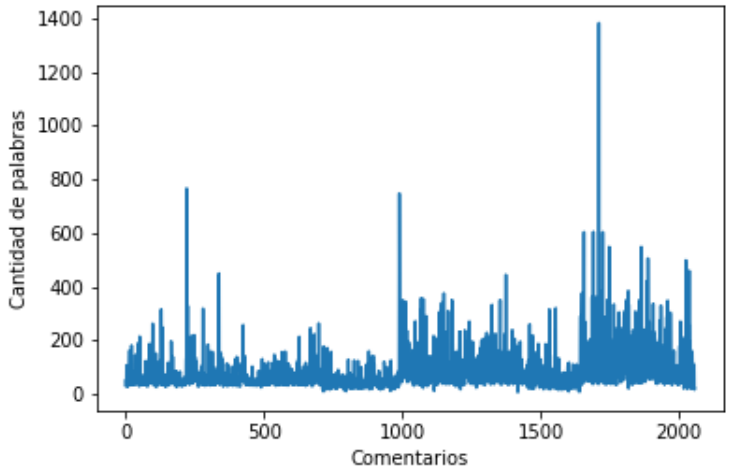

Figura 4: Distribución de objetivos de los comentarios (Elaboración propia).

En la Figura 4 se puede ver una gráfica de la cantidad de palabras de cada uno de los comentarios del *corpus*. Queda demostrado que la longitud de los elementos es indefinida.

3.3.4. Homogeneidad

El tener un conjunto de ejemplos que son similares entre ellas ayudan a interpretar y sacar mejores conclusiones de ellos (Sinclair y Wynne, 2005), pero ¿cómo se determina que dos ejemplos son similares? En análisis de sentimientos se puede decir que dos opiniones que tienen el mismo objetivo, fecha, idioma, entorno u otros, son similares o que ambas tienen la misma interpretación.

La homogeneidad es el atributo que se refiere al grado de similaridad en un conjunto de datos. Para el caso del *corpus* conformado si todos los textos fueran similares tanto que expresan prácticamente lo mismo, este sería inútil. La similaridad a la que el *corpus* debería

apuntar esta en un nivel muy bajo, pero lo suficiente para que los elementos pertenezcan a un subconjunto específico de una población.

Al estar tratando con conjuntos de textos, se busca que todos pertenezcan a un tema y entorno similar. Pero dependiendo del recolector pueden existir problemas de interpretación debido a diferencias semánticas por el contexto o vocabulario. Aunque esto se puede solucionar definiendo un vocabulario aceptado, la variedad de palabras del *corpus* es un factor valioso para la generalización.

La similaridad entre los comentarios que se busca se logrará utilizando los criterios de selección antes mencionados, ya que así se conformará un espectro de comentarios que pertenezcan al tema de interés.

3.4. CALIDAD DE LOS DATOS

Los datos son cruciales para el proceso de aprendizaje de una red neuronal, ya que la calidad de ellos determina en gran medida su desempeño pero ¿qué factores determinan la calidad de los datos? En las secciones anteriores se ha estado definiendo algunas propiedades que ayudarán a calificar el *corpus*, estas son la representatividad, balance, longitud y homogeneidad.

Para medir el *corpus* usando estas propiedades se podrían diseñar técnicas que calculen con números el cumplimiento de tales cualidades, pero eso está fuera del alcance de este trabajo. En su lugar se calificó de una manera más interpretativa que resultó en analizar el *corpus* manualmente y determinar que tan bien se cumplen las propiedades.

Sobre la representatividad, el tamaño objetivo se eligió teniendo en cuenta un número de datos lo suficientemente grande para alimentar una red neuronal de convolución. De acuerdo a las pruebas realizadas durante el desarrollo del trabajo se determinó que 1000 datos serían suficientes para entrenar una red pequeña pero aceptable. En la Tabla 3 se muestra cómo se cumple la propiedad de representatividad y también de balance, ya que la cantidad de comentarios que cae en cada etiqueta es la misma.

Etiqueta / Sentimiento	**Cantidad de comentarios**
Negativo	400
Neutral	400
Positivo	400
Total	1200

Tabla 3: Balance de la cantidad de comentarios recolectados.

Cabe mencionar que el objetivo de comentarios a alcanzar se determinó con el tiempo, ya que como se explicará en el Capítulo 4 la recolección e implementación del modelo fue un proceso iterativo, por lo que las primeras versiones del modelo contaba con 100 o 200 comentarios por etiqueta.

Respecto a la homogeneidad, tal cual explicado en la sección anterior, se dirá que se cumple esta propiedad si los criterios de selección antes definidos son usados para recolectar el *corpus* y de hecho así fue. Los criterios de selección son los primeros en influir en la calidad del *corpus* ya que determinan un tema no dependiente del vocabulario.

Se va a posponer la explicación del proceso realizado para solventar el problema de la longitud de los comentarios para las siguientes secciones. Por ahora se dirá que se cumple con la propiedad de longitud al lograr tener comentarios con un tamaño fijo de 100 palabras antes de utilizar algún algoritmo de *Machine learning*.

Se sabe que el conjunto de datos no es perfecto pero si se tratara de solventar cada uno de esas preocupaciones se pecaría de perfeccionismo. Cuando se crea un *corpus* los objetivos de representatividad y longitud se deben tener en cuenta, pero debido a que son metas que dependen de las circunstancias basta con que sean usadas para guiar el diseño de un *corpus* (Sinclair y Wynne, 2005).

Entre los problemas que se pueden mencionar, están el desbalance entre los objetivos de los comentarios. Las opiniones dirigidas a hoteles tienen gran predominancia sobre el resto. Otro problema es la ortografía que no se tuvo en cuenta durante la planificación, algunos comentarios pueden verse afectados pero es algo que se podría solucionar durante el preprocesamiento. La longitud de las opiniones es variable, pero esta es una preocupación menor por ahora.

La inserción de comentarios inventados y traducidos debería ser de bajo riesgo ya que se tuvo el cuidado de que sean similares a comentarios reales, es decir homogéneas. Además el impacto es mínimo ya que menos del 5 % del *corpus* tiene este origen.

3.5. PREPROCESAMIENTO

Esta etapa tiene como objetivo liberar a los textos crudos de cualquier imperfección no deseada, como ortografía, vocabulario o longitud. Para los cuales se cuentan con algunas técnicas pero su uso depende de la aplicación.

Como primer paso se podría pensar en remover caracteres extraños del *corpus*, aquellos

que no aportan ningún valor o que podrían traer resultados extraños en el futuro. Otras técnicas conocidas dentro del procesamiento de lenguaje natural son *stemming* y palabras de escape.

Por razones gramaticales las palabras tienen distintas formas, tales como *trabajo* y *trabajando*. En diversas aplicaciones suele ser común querer que ambas palabras tengan el mismo significado, pero el problema yace en que ambos tengan valores asociados diferentes debido a derivaciones gramaticales. La técnica de ***stemming***, en términos simples, logra obtener la raíz de una palabra recortando los caracteres que derivan. Por ejemplo con la palabra *trabajo*, luego de pasar por un algoritmo de *stemming* resulta en "*trabaj*" (Manning, Raghavan, y Schütze, 2008).

Este método en realidad tiene mayor utilidad en idiomas como inglés donde las palabras comúnmente tienen tres variaciones, por ejemplo *organize*, *organizes* y *organizes*. Aún así, existen implementaciones que pueden aplicarse a otros idiomas como el español, pero no será necesario en este trabajo por el uso de *word embeddings* explicado más adelante.

Las **palabras de escape** tienen un uso más universal ya que solucionan el dilema de las palabras más frecuentes en un texto. La predominancia de ciertos términos tales como artículos y conjunciones ocasionan que los modelos se inclinen a aprender con más empeño estas palabras que otras más importantes pero menos frecuentes. Las palabras de escape son el conjunto de los términos más frecuentes de un idioma que deben ser removidos si son encontrados en las filas del *corpus*, o simplemente ignorarlos. No se consideró hacer uso de este método por cuenta propia ya que es algo se zanjó de otra manera.

Durante esta etapa si se realizó un proceso de truncado sobre la longitud de un comentario. Para normalizar el tamaño variante de los ejemplos del *corpus*, se selecciona un tamaño máximo y se trunca cada ejemplo. La longitud de un comentario se mide por el número de palabras, ya que el tamaño de una palabra es trivial por el uso de *word embeddings*. Si el comentario tiene más palabras que el máximo esta es cortada, si es menor se rellenan los espacios con ceros. La pérdida de información es inevitable pero se trató que sea mínima al elegir un tamaño que afecte a menos comentarios como se pudo. Esto no debería verse como un incumplimiento a la propiedad de longitud ya que el objetivo es que el método de *Machine learning* que generalice el *corpus* no pierda información durante el entrenamiento, la solución es controlar la pérdida antes de que esto suceda.

Aparte de encargarse del tamaño, el preprocesamiento del *corpus* terminó siendo sencillo y minimalista. Es suficiente convertir los textos a minúscula y remover caracteres extraños del mismo. El uso de ***word embedding*** es requerido para crear las características que alimentarán

la red neuronal, pero su uso también tiene otras ventajas. Utilizar *word embeddings* alivia deficiencias que conjuntos de datos pequeños y con ruido puedan tener.

3.5.1. Palabras embebidas

Las palabras embebidas o *word embeddings* son un método de representación de palabras en un espacio n-dimensional que pueden ser usadas como características para alimentar modelos de *Machine learning* o *Deep learning*. Por si solos los *word embeddings* tienen propiedades interesantes y pueden ser usadas en solitario para realizar análisis de datos.

Entre las propiedades de los *word embeddings*, la más importante debe ser el espacio vectorial que conforman y la relación entre los vectores de cada palabra. La cercanía entre dos vectores indica su similitud, esto es, cuanto la distancia entre dos vectores de palabras es más cercana a cero significa que las palabras correspondientes son más similares en significado. Es posible realizar analogías, utilizando simplemente *word embeddings* y una forma de medir la distancia, para lo cual los candidatos son la distancia por cosenos o diferencias cuadráticas.

La idea detrás de su composición es entrenar un modelo con cantidades enormes de textos, lo que significa que es un proceso lento. Uno de los modelos de *word embeddings* más populares y sencillos en los últimos años es el llamado *skip gram* propuesto por Mikolov, Sutskever, Chen, Corrado, y Dean (2013) que consiste en predecir palabras objetivo dado una palabra como contexto. Las muestras se generan a partir de los textos iniciales, donde una palabra es escogida al azar y luego, por ejemplo, las 4 palabras más cercanas a la izquierda o derecha son sus objetivos. Otro modelo también propuesto por Mikolov y cols. (2013) toma en cuenta varias palabras como contexto para predecir un objetivo, este método es conocido como *cbow* (*Continuous Bag of Words*).

Muchos suelen decir que los *word embeddings* se generan con la ayuda de aprendizaje no supervisado ya que no se cuenta datos etiquetados y pueden tener razón con algunas propuestas. Pero en el caso de *skip gram* y *cbow* los textos iniciales son procesados para generar ejemplos etiquetados y así solucionar la cuestión con métodos de aprendizaje supervisado. Los *word embeddings* ya tienen muchos años de existencia además de tener el registro histórico de haber empezado con modelos complejos que tardaban meses en entrenar como el de Collobert y cols. (2011) hasta llegar a algoritmos más sencillos tales como *skip gram* que dependiendo del tamaño del vocabulario pueden tardar desde horas hasta semanas.

En la actualidad los modelos de *word embeddings* comerciales son entrenados con

documentos y vocabularios de tamaños entre millones y billones de palabras. Pero para fines académicos, un vocabulario de 1000000 palabras sería suficiente.

3.5.2. Transferencia de aprendizaje

La transferencia de aprendizaje es un término acuñado por la investigación de *Machine learning* y *Deep learning* que tiene como objetivo prestar o transferir el conocimiento aprendido por una tarea A a una tarea B para que mejore su aprendizaje (Torrey y Shavlik, 2009). Tales como permitir aprender funciones más complejas teniendo poca información inicial, o abarcar dominios más extensos gracias a la herencia de un modelo ya entrenado.

En esta investigación el uso de *word embeddings* tiene varios motivos y la transferencia de aprendizaje es uno de ellos. Con tal de mejorar el entrenamiento del modelo a presentar se quiere aprovechar al máximo los beneficios de un modelo pre-entrenado y solventar dos preocupaciones, la limitada cantidad de comentarios y el mapeo de textos a tensores (que pueden ser utilizados para entrenar una red neuronal).

3.5.3. Mapeo

El *word embedding* que se utilizó fue entrenado por Cardellino (2016), con la herramienta *word2vec* la cual es una implementación de los algoritmos propuestos por Mikolov y cols. (Mikolov y cols., 2013). Fue entrenado con cerca de 1.5 billones de palabras con textos de la web, quitando las 273 palabras más comunes del idioma obviando la necesidad de realizar preprocesamiento por palabras de escape.

Ahora que ya se tiene el *word embedding* lo siguiente es realizar el mapeo de los comentarios en datos que realmente pueden ser usados por una red neuronal convolucional.

Cada comentario es mapeado en un tensor (matriz) que es conformada apilando los vectores de cada palabra. Para entrenar una red neuronal convolucional cada matriz debe tener el mismo tamaño, el ancho es igual a la dimensión del vector d, la cual de acuerdo al *word embedding* de Cardellino (2016) es 300. El alto n corresponde a la cantidad de palabras máximas que se soporta por oración o comentario, este es un hiperparámetro del modelo, mencionado antes en el truncado los comentarios ($n = 100$).

Entrando en los detalles del mapeo, un *word embedding* funciona como una estructura de datos parecida a un diccionario o tabla-hash, donde se proporciona una llave para obtener el valor que guarda. La llave y la palabra en cuestión deben coincidir perfectamente con tal

de obtener el vector correspondiente. Se pueden elegir el conjunto de las llaves a usar de antemano o simplemente usar todas las palabras que aparezcan en los datos, eso depende del creador del *word embedding*. Debido al uso de un *word embedding* pre-entrenado se está obligado a usar el mismo vocabulario, razón por la que no se utilizó *stemming* en el preprocesamiento, ya que se corre el riesgo de que las palabras dejen de funcionar como llave.

La creación del *word embedding* no utilizo un vocabulario predefinido, todas las palabras existentes en los textos que conforman su conjunto de entrenamiento son las llaves. Debido a las condiciones de creación del *word embedding*, este es tolerante a diferentes variaciones de una palabra como plurales, con acentos, errores ortográficos, o incluso en inglés.

3.6. DIVISIÓN DE LOS DATOS

Los proyectos de *Machine learning* suelen tener al menos dos conjuntos de datos, uno de **entrenamiento** y otro de **pruebas**, los cuales son usados para el aprendizaje y evaluación del modelo respectivamente. Las proporciones de tamaño suelen ser de 80 % para los datos entrenamiento y los restantes 20 % para pruebas.

En fin, la Tabla 4 indica la configuración usada para dividir el *corpus*, cabe recalcar que los datos fueron divididos aleatoriamente manteniendo un balance entre la polaridad de los comentarios que entran en cada subconjunto.

Es importante que el conjunto de pruebas no sea utilizado para entrenar o ajustar hiperparámetros del modelo ya que no se podría confiar en los resultados obtenidos. Si es que los datos de prueba llegan a usarse para ajustar detalles del diseño durante el entrenamiento, podría provocarse un problema de varianza alta[3].

Conjunto de datos	**Porción del *corpus***
Entrenamiento	80 %
Validación	10 %
Pruebas	10 %

Tabla 4: División de los datos.

Es por eso que se suele tener otro conjunto llamado de **validación** o desarrollo, el cual puede ser usado para hacer los ajustes necesarios en el entrenamiento y desde que el conjunto de validación y de pruebas son diferentes se evita la auto introducción de varianza alta en el

[3]El modelo se ajusta demasiado bien a los datos de entrenamiento, tanto que predice mal sobre ejemplos nuevos

modelo. El conjunto de desarrollo también es usado para técnicas de validación cruzada, pero debido al pequeño tamaño del *corpus* se hubiera perdido representatividad si se intentaba.

Métodos como *fold cross-validation* dividen el conjunto de entrenamiento en varios lotes, y se entrenan tantos modelos como lotes de datos pero cada una con diferentes hiperparámetros. Al terminar se determina cuál fue la configuración que funciona mejor y se dice que esos valores son los elegidos (Goodfellow y cols., 2016).

CAPÍTULO 4 DISEÑO E IMPLEMENTACIÓN DE LA RED

Cuando se habla del diseño de una red neuronal se refiere a la forma y composición interna de esta, aunque las decisiones tomadas sobre los conjuntos de datos también son parte del diseño. Para diferenciarlos, a la composición interna de una red neuronal se conocerá mejor como arquitectura.

En el capítulo anterior ya se explicaron las propiedades, procesamiento y división de los datos, por lo que en este capítulo se tocará el algoritmo de entrenamiento y la arquitectura de la red neuronal base que se ha desarrollado para probar la hipótesis de esta investigación.

4.1. FUNCIONES DE ACTIVACIÓN

Una función de activación es simplemente una función matemática no lineal.

Como ya se mencionó en el Capítulo 2, las funciones de activación tienen un rol muy importante en la definición de redes neuronales. Son las encargadas de introducir una no-linealidad, por lo que se suele entenderlas como cerrojos que controlan el flujo de activaciones de una red, similar a las neuronas de un cerebro biológico donde se disparan impulsos eléctricos.

Para un perceptrón, clasificador binario, es suficiente contar con una función de activación umbral que devuelve 1 si la entrada es mayor o igual a cierto valor (el umbral) y 0 o -1 si es menor. Pero cuando se trata de una red neuronal donde agrupaciones de perceptrones están conectadas entre sí, tales valores enteros no son propicios para la propagación ya que el aprendizaje se ve mermado por la poca información que se transmite. Aún así, esta función umbral es útil para clasificaciones binarias y puede ser usada como función de activación en la última capa de una red neuronal para tareas de clasificación.

La primera función de activación no lineal que se mencionó en el Capítulo 2 fue la

sigmoide, la cual tiene una salida acotada entre 0 y 1. Similar a la función umbral pero más apropiada para entrenar redes neuronales ya que retorna información probabilística, útil para las capas siguientes.

$$sigmoide(z) = \frac{1}{1+e^{-z}}$$

Pero su uso suele provocar un problema llamado **desvanecimiento del gradiente**, el cual es principalmente causado por entrenar una red neuronal con muchas capas de profundidad. Con el tiempo los valores del gradiente llegan a ser cercanos a 0 y el aprendizaje puede ralentizarse demasiado, tanto que la red se rehúsa a entrenar. La *sigmoide* no ayuda con este problema ya que su derivada ronda entre 0 y 0.25 (muy cercano a 0). Pero aún así es popular y útil en redes redes neuronales poco profundas (Goodfellow y cols., 2016).

Otra función de activación no lineal popular es la llamada tangente hiperbólica ***tanh***, la cual aún sufre del problema de desvanecimiento de gradiente pero tiene un rango más grande y menos posibilidades de ser 0. Esta función es ampliamente usada al igual que la *sigmoide* con la ventaja que el gradiente generado es más fuerte y no se desvanece tan rápido, su fórmula es la siguiente.

$$tanh(z) = \frac{2}{1+e^{-2z}} - 1$$

La fórmula puede resultar familiar, y es que *tanh* es la función *sigmoide* escalada como se puede ver a continuación.

$$tanh(z) = 2\ sigmoide(2z) - 1$$

En la Figura 5 se puede ver una gráfica de la *sigmoide* donde se distingue claramente que la salida de esta función varía entre 0 y 1. La función de activación *tanh* tiene un gráfico similar con la diferencia que su salida en y tiene un rango más amplio, entre -1 y 1.

En las redes neuronales modernas la función de activación elegida por defecto es la función *ReLU* (*Rectified Linear Unit*) definida por la siguiente fórmula.

$$relu(z) = max(0, z)$$

A pesar de su apariencia en la Figura 5 es no lineal, pero posee propiedades de una función lineal que son ventajosas para el entrenamiento de redes neuronales. Ventajas como el bajo costo computacional y una derivada sencilla de calcular que incluso suele acelerar el aprendizaje.

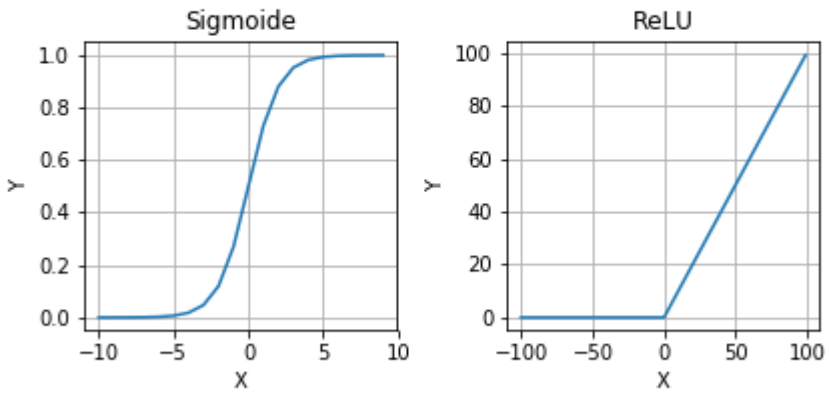

Figura 5: Funciones de activación (Elaboración propia).

Un lado negativo de la función *ReLU*, es que una red neuronal no puede aprender cuando la activación de un ejemplo resulta ser 0, pero esto se puede solucionar de diversas maneras. La más común es usar una variación de *ReLU* llamada ***Casi ReLU***[1] que está definida por,

$$relu(z) = max(\alpha_i\, z, z)$$

donde α_i es un valor muy pequeño. Logrando así que el lado negativo de la función *ReLU* no sea totalmente paralela al eje X (ver Figura 5) sino que tenga una ligera inclinación hacia abajo, lo suficiente para solventar el problema antes mencionado (Goodfellow y cols., 2016). En próximas secciones cuando se mencione la función *ReLU* se referirá directamente a *Casi ReLU*.

Por último se presenta la función *softmax*, la cual se caracteriza por calcular probabilidades en una distribución de valores. Puede verse a esta función como una generalización de *sigmoide*, la cual calcula probabilidades en una distribución binaria.

Formalmente la función *softmax* está definida por,

$$softmax(z_i) = \frac{e_i^z}{\sum_j^n e^{z_j}}$$

para la salida de la neurona i, el divisor se evalúa con todos los valores de la activación anterior el cual es el mismo para cada neurona.

La función *softmax* es usada comúnmente como activación en la última capa de una red neuronal, para realizar la predicción de una etiqueta entre un conjunto finito de ellas. Es muy utilizada en tareas de clasificación y será utilizada en la red neuronal convolucional de esta

[1] El término original en inglés es *Leaky ReLU*, que será traducido como *Casi ReLU*.

investigación también.

4.2. REGULARIZACIÓN

Uno de los problemas centrales de *Deep learning* es lograr que un modelo funcione bien no solo con los datos de entrenamiento sino también sobre nuevos. Con eso en mente se han diseñado muchas estrategias que reducen el error sobre el conjunto de pruebas a costa de incrementar el error durante el entrenamiento. Estás técnicas son conocidas colectivamente como regularización.

La **Regularización** es cualquier modificación hecha a un algoritmo de entrenamiento con la intención de reducir su error de generalización, no su error de entrenamiento (Goodfellow y cols., 2016). Eso quiere decir que es preferible sacrificar el desempeño sobre el conjunto de entrenamiento con tal de obtener mejores resultados sobre datos nuevos, el conjunto de pruebas por ejemplo.

A lo largo de este documento se ha mencionado bastante a los conjuntos de entrenamiento y de prueba pero no así al conjunto de validación. Recordar que en el Capítulo 3 ya se habló sobre los conjuntos de datos que se usarían y como se dividen. Este conjunto de validación, también conocido como de desarrollo, se utiliza para ajustar el modelo durante las etapas de diseño y entrenamiento de la red neuronal.

El conjunto de validación es usado para identificar problemas de varianza alta o sobre entrenamiento. La varianza es un concepto probabilístico que indica el grado de dispersión de una distribución, evaluados con respecto a una función que intenta generalizar tal distribución. Se determina que un modelo sufre de sobre entrenamiento si su desempeño es pobre sobre el conjunto de validación.

Existen diversos métodos de regularización aplicables a *Deep learning*, pero se explicarán las dos técnicas utilizadas en este trabajo. Una se conoce como ***Dropout*** y es exclusiva de redes neuronales. La segunda se llama **Parada temprana**, la cual es una modificación al algoritmo de entrenamiento en sí, por lo que es aplicable a otras técnicas fuera de de *Deep learning*.

El método de *dropout* en teoría consiste en agrupar unidades lógicas (neuronas) de una red neuronal para así entrenar un diseño aislado y completamente diferente del original. Este proceso termina con el entrenamiento de varias redes neuronales, cada una independiente y con su propio dominio de datos. Cada red entrenada termina siendo especialista en características más específicas de un subconjunto de datos, evitando el sobre entrenamiento

de la red (Goodfellow y cols., 2016).

En la práctica, entrenar tantas redes diferentes es ineficiente y tardío, por ello *dropout* en realidad se implementa como una máscara. Durante el entrenamiento, se interceptan los resultados de capas anteriores y se aplica multiplicación de matrices para convertir en 0 las activaciones de ciertas unidades. Esto da la sensación de entrenar un diseño diferente en cada iteración, logrando el objetivo. Así las neuronas aprenden parámetros menos dependientes del resto porque estas creen que las demás podrían desaparecer en cualquier momento. *Dropout* utiliza un valor llamado **tasa de probabilidad** para saber el porcentaje de cuántas neuronas se deben ocultar cada vez.

El segundo método de regularización mencionado, la parada temprana, es una técnica que consiste en detener el entrenamiento cuando se nota que el error del conjunto de validación está creciendo (Goodfellow y cols., 2016). Se implementa dentro el algoritmo de aprendizaje como un alto en seco ignorando el número de iteraciones restantes. Se tocarán los detalles en la siguiente sección.

Aparte de las técnicas de regularización que se usarán, cuando se trata de redes neuronales convolucionales un método de regularización muy popular en tareas de visión por computadora es el **Aumento de datos**. Esto consiste en incrementar la cantidad de los ejemplos en el conjunto de entrenamiento, validación y pruebas realizando ciertas operaciones sobre las imágenes. Operaciones como rotar, desenfocar, cortar y acercar. Pero al tratarse esta investigación de un problema perteneciente al campo de procesamiento de lenguaje natural, no existen operaciones tan sencillas que permitan generar textos nuevos y válidos. Un enfoque planteado por Zhang, Zhao, y LeCun (2015) consitió en utilizar la web **www.thesaurus.com** para generar oraciones similares con la ayuda de sinónimos, pero los resultados no fueron lo suficientemente buenos.

4.3. ALGORITMO DE APRENDIZAJE

Es posible entrenar la red neuronal convolucional a presentar con el algoritmo de descenso de gradiente explicado en el Capítulo 2, pero este algoritmo podría no ser el más adecuado ya que presenta deficiencias con modelos complejos. Una red neuronal convolucional es un tipo de red que aprende funciones muy complejas y lo hace demasiado rápido con conjuntos de entrenamiento pequeños. Para asegurar su éxito es necesario entrenar una red de convolución con cantidades de datos que van de miles a millones.

Un algoritmo de entrenamiento como el descenso de gradiente depende de todos los

datos en cada iteración que realiza, este proceso es increíblemente costoso provocando que el tiempo de entrenamiento crezca con la cantidad de datos. Para solventar la falta de eficiencia del algoritmo de descenso de gradiente, existe un algoritmo que es más veloz sin disminuir la efectividad del entrenamiento. Este algoritmo es conocido como **descenso de gradiente estocástico**; *SGD* por sus siglas en inglés.

El descenso de gradiente estocástico, tiene el mismo esqueleto que su predecesora. Calcular el gradiente y actualizar los parámetros en cada iteración, la diferencia radica en el conjunto de datos utilizado para actualizar los parámetros cada vez. El algoritmo *SGD* tiene un ciclo iterativo exterior al igual que descenso de gradiente, pero dentro dicho ciclo se realiza otro procedimiento previo a calcular el gradiente y actualizar los parámetros. Se pueden ver los detalles en el Algoritmo 2.

El problema de depender de todos los ejemplos en el conjunto de entrenamiento se solventa dividiendo tal conjunto en agrupaciones más pequeñas llamadas **mini lotes**. Cada mini lote es creado eligiendo aleatoriamente una cantidad de ejemplos del conjunto de entrenamiento. En *SGD* se itera N veces, en cada iteración primero se divide el conjunto de datos en mini lotes y luego se realiza la actualización de parámetros con cada uno (Goodfellow y cols., 2016).

Algoritmo 2 Descenso de gradiente estocástico

Input Tasa de aprendizaje (α)
Input Número de iteraciones (N)
Input Tamaño de minilote (k)
Input Conjunto de entrenamiento (X, Y)
Output Parámetros entrenados (θ)

$\theta \leftarrow inicializar\_parametros()$
for N iteraciones **do**
  $minilotes \leftarrow generar\_minilotes(X, Y, k)$
  for each (X_i, Y_i) en $minilotes$ **do**
    $A \leftarrow propagacion\_hacia\_adelante(\theta, X_i)$
    $\hat{g} \leftarrow propagacion\_hacia\_atras(\theta, Y_i, A)$
    $\theta \leftarrow \theta - \alpha\, \hat{g}$
  end for
end for
returns θ

Es importante que los mini lotes sean creados aleatoriamente, de otro modo el algoritmo

de entrenamiento deja de funcionar. Se debe romper la simetría, para que la red aprenda funciones no lineales de más alto orden. Una red profunda aprende funciones más complejas porque la dimensionalidad de ella aumenta junto con la profundidad.

Este método de entrenamiento tan poco dependiente de los datos en la actualización de parámetros puede dar la sensación de que no funciona, pero la verdad es que con suficientes iteraciones y grandes cantidades de datos los resultados son igualables al descenso de gradiente. Con este algoritmo, se puede ver que en las gráficas de curva de aprendizaje el error crece y decrece bruscamente entre iteraciones esto es normal y en el siguiente capítulo se podrán ver ejemplos de esta afirmación.

En esta investigación se hace uso del algoritmo de descenso de gradiente estocástico para entrenar la red neuronal convolucional. A medida que avanzaba el desarrollo se probaron también con diferentes arquitecturas y métodos de regularización. Uno de los métodos de regularización añadidos en la experimentación es la **parada temprana**. En el Algoritmo 3 se detalla la implementación de esta técnica.

La parada temprana, como ya se mencionó, consiste en detener el entrenamiento antes que el error sobre el conjunto de validación, o de desarrollo, comience a incrementarse. Este proceso en realidad puede hacerse manualmente, estudiando el comportamiento de la curva de aprendizaje y el desempeño, pero es más efectivo y apropiado realizar la parada temprana de manera automática. Es necesario adaptar el algoritmo de *SGD* para detener el entrenamiento cuando sea el momento.

Como se ve en el Algoritmo 3, ahora existe un control que detiene el entrenamiento cuando el error sobre el conjunto de validación empieza a elevarse. El algoritmo termina cuando el error estuvo subiendo por un número i de iteraciones, tal que i es mayor que la paciencia P. Si el error deja de subir y no se ha sobrepasado la paciencia, es un falso positivo y el entrenamiento continúa hasta llegar a las N iteraciones o superar la paciencia P.

En los algoritmos presentados se puede ver una función que inicializa los parámetros (θ) nombrada $inicializar\_parametros()$. Esta función abstrae el método de inicialización de los tensores que se asignan a la variable θ. Para romper la simetría y entrenar efectivamente una red neuronal, los tensores iniciales no pueden ser inicializados con todos sus elementos en 0. Normalmente se utilizan valores aleatorios o algún método de distribución que acelera el entrenamiento en sus primeras iteraciones. En este trabajo se ha implementado la inicialización con la ayuda de la distribución normal truncada (Goodfellow y cols., 2016).

Algoritmo 3 Descenso de gradiente estocástico + Parada temprana

Input Tasa de aprendizaje (α)
Input Número de iteraciones (N)
Input Número de iteraciones entre evaluaciones (n)
Input Tamaño de minilote (k)
Input Paciencia (P)
Input Conjunto de entrenamiento (X, Y)
Input Conjunto de validación (X_{val}, Y_{val})
Output Parámetros entrenados (θ)
Output Número de iteraciones que tomo entrenar (i)

$\theta \leftarrow inicializar\_parametros()$
$i \leftarrow 0$
$iteraciones\_errando \leftarrow 0$
$error\_validacion \leftarrow \infty$
while $i < N$ **and** $iteraciones\_errando < P$ **do**
  for n iteraciones **do**
    $minilotes \leftarrow generar\_minilotes(X, Y, k)$
    for each (X_i, Y_i) en $minilotes$ **do**
      $A \leftarrow propagacion\_hacia\_adelante(\theta, X_i)$
      $\hat{g} \leftarrow propagacion\_hacia\_atras(\theta, Y_i, A)$
      $\theta \leftarrow \theta - \alpha\, \hat{g}$
    end for
  end for
  $i \leftarrow i + n$
  $error\_val\_actual \leftarrow error\_validacion(\theta, X_{val}, Y_{val})$
  if $error\_val\_actual < error\_validacion$ **then**
    $iteraciones\_errando \leftarrow 0$
    $error\_validacion \leftarrow error\_val\_actual$
  else
    $iteraciones\_errando \leftarrow iteraciones\_errando + 1$
  end if
end while
returns θ, i

Se puede notar que el algoritmo *SGD* + Parada Temprana, introduce nuevos hiperparámetros a ajustar. La paciencia *P*, indica la cantidad máxima de iteraciones a tolerar

cuando el error de validación sube. El tamaño del mini lote k suele ser un valor potencia de 2, como ser 16, 32, 64, etc. El número de iteraciones para evaluar el error n se refiere a cuántas veces se actualizarán los parámetros antes de calcular el error de validación.

En los últimos años la mayoría de los modelos de *Deep learning* suelen ser entrenados con *SGD*, debido a su menor costo temporal, en comparación a su predecesor el descenso de gradiente. Como se mencionó en su momento el algoritmo de descenso de gradiente también suele llevar el nombre de "por lotes" porque en cada iteración se procesan todos los ejemplos del conjunto de entrenamiento. Es muy ineficaz para entrenar redes que necesitan conjuntos de datos muy grandes. Aquí es donde entra el algoritmo *SGD* al rescate ya que soluciona esta deficiencia.

4.4. ARQUITECTURA

En teoría, las redes neuronales pueden aproximar cualquier función, es por eso que son conocidas como aproximadores universales (Goodfellow y cols., 2016). Es solo cuestión de encontrar la arquitectura ideal que resuelva correctamente el problema que se tiene. Un proyecto de *Deep learning* puede llevar meses de trabajo probando diferentes arquitecturas antes de toparse con un diseño prometedor. Para acelerar las cosas, es frecuente iniciar un proyecto teniendo como base una arquitectura que soluciona un problema similar. Una arquitectura que se ha probado funciona para clasificación de oraciones en procesamiento de lenguaje natural puede funcionar para análisis de sentimiento (otra tarea de clasificación en procesamiento de lenguaje natural).

La arquitectura base y uno de los diseños que inspiró esta investigación es la red neuronal convolucional planteada por Kim (2014), donde se presenta una red de convolución relativamente sencilla que puede ser utilizada para diversas tareas de clasificación de texto. Ha tenido resultados muy positivos con *corpus* originarios de *Twitter* para tareas de clasificación como análisis de sentimientos, los resultados incluso alcanzan la precisión del estado del arte.

El diseño de Kim consiste en introducir oraciones, transformadas a matrices, en una red neuronal que tan solo tiene una capa de convolución para luego predecir la etiqueta con una salida *softmax*. La matriz de entrada está conformada por una oración, donde cada fila representa una palabra. Cada vector fila se obtiene mapeando la palabra actual contra un *word embedding* pre-entrenado. La matriz tiene una dimensión fija, el ancho (w) es la dimensión del *word embedding* (normalmente 300) y el alto (h) es un hiperparámetro constante. Si una

oración contiene más palabras que el alto esta es cortada, por otro lado si tiene menos se rellena con ceros hasta completar la matriz.

La primera capa de convolución consta de una agrupación de F filtros, cada uno con altura f_i y ancho w. En la arquitectura de Kim, se hace uso de los denominados **filtros anchos**, llamados así por su forma (tienen el mismo ancho que el mapa de características w). Con un filtro ancho, la convolución se simplifica y solo se tiene una ventana que se desplaza en una dimensión (de arriba hacia abajo) tomando en cuenta el tamaño del paso s. Esto genera un tensor resultante con dimensión $(h \times 1)$.

La convolución se ejecuta sin relleno ($p = 0$) y con paso de uno ($s = 1$). El alto de filtros seleccionados por Kim son 3, 4 y 5, con alrededor de 100 filtros para cada uno. En la etapa de detector se utiliza la función $ReLU$ como activación, aunque también experimentó con *tanh*.

En la tercera etapa de la convolución, Kim utiliza la función *max-pooling* sobre el resultado de cada convolución. La dimensión de la ventana del *pooling* es igual al resultado de la convolución, por lo que un valor de activación es recuperado por cada filtro que se tenga. Al obtener un único resultado por cada *pooling* es posible tener filtros con un alto diferente (recordar que todos los filtros tienen un ancho igual a la dimensión del *word embedding*) en una misma capa ya que los resultados son concatenados. Todas las activaciones de *pooling* se apilan en un único vector resultado que luego se propaga a la siguiente capa.

Como se ve en la Figura 6 Kim utiliza *dropout* como método de regularización en la última capa la red, la cual contiene la activación *softmax*. Con una **tasa de probabilidad de *dropout*** igual a 0.5, se indica que la mitad de las activaciones serán ocultadas en cada iteración. También se puede ver que el tensor de entrada tiene dos canales, esta es una opción que se experimenta en Kim (2014) pero que no se explorará en este trabajo.

La última capa debe tener tantas unidades de salida como cantidad de etiquetas tenga el *corpus*. La función de activación *softmax* es apropiada para este tipo de tarea por su salida probabilística.

Como ya se mencionó, la arquitectura implementada en este trabajo toma inspiración de Kim (2014) en su mayoria, pero Hughes y cols. (2017) también tuvo un gran impacto. Hughes plantea una red más profunda, con dos capas contiguas de convolución al inicio y capas de red completamente conectadas luego. Esta red fue utilizada para clasificar documentos médicos entre 16 etiquetas posibles, con el propósito de servir para la inteligencia artificial Watson. En un principio se quiso que la red de esta investigación siga el ejemplo de Hughes, pero un *corpus* con un tamaño tan limitado como el que se tiene no es suficiente para un diseño tan

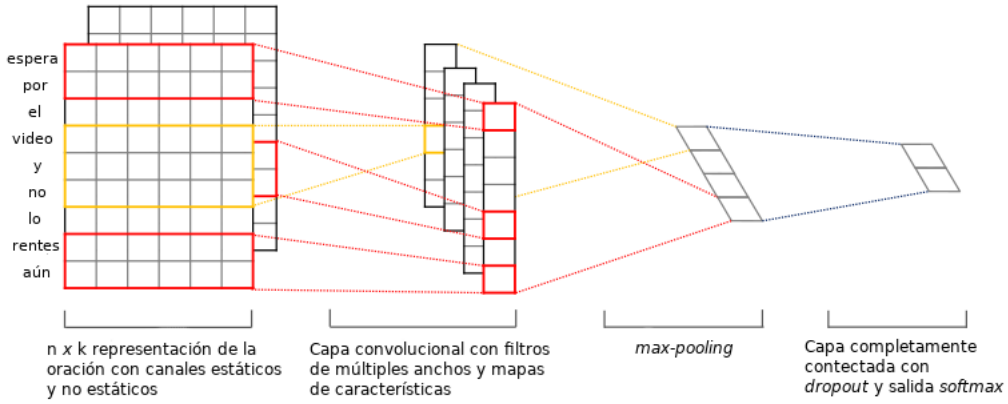

Figura 6: Arquitectura de la red neuronal convolucional de Kim con un ejemplo de dos canales (Adaptado de Yoon Kim 2014).

complejo.

En este trabajo se probaron diferentes configuraciones y variaciones de la arquitectura base, los resultados de estos intentos realizados durante la experimentación se detallarán en el Capítulo 5. En esta sección no se presentarán todos los diseños probados, en su lugar se explicará una arquitectura general que representa la evolución del diseño.

En la Figura 7 se muestra la arquitectura de red neuronal convolucional desarrollada en este trabajo. La entrada es un tensor bidimensional (una matriz) con alto h y ancho w. El ancho corresponde a la dimensión de los *word embeddings* utilizados para componer la entrada, proceso explicado en el Capítulo 3.

El diseño presentado consta de tres capas, convolución, completamente conectada y *softmax*. A diferencia de la arquitectura de Kim hay una capa adicional entre la salida y la capa de convolución.

Las siguientes fórmulas explican mejor lo que ocurre dentro de la capa de convolución. Recordar la notación introducida en el Capítulo 2, f y b son el filtro y bias respectivamente (los parámetros). El primer superíndice que se puede ver en f^{ij} indica la capa y el segundo es el número de unidad en dicha capa. Z representa un resultado intermedio entre la activación lineal y *pooling*.

$$Z^{11} = f^{11} * X + b^{11} \ , \ \ A^{11} = max\_pool(relu(Z^{11}))$$

$$Z^{12} = f^{12} * X + b^{12} \ , \ \ A^{12} = max\_pool(relu(Z^{12}))$$

$$Z^{13} = f^{13} * X + b^{13} \ , \ \ A^{13} = max\_pool(relu(Z^{13}))$$

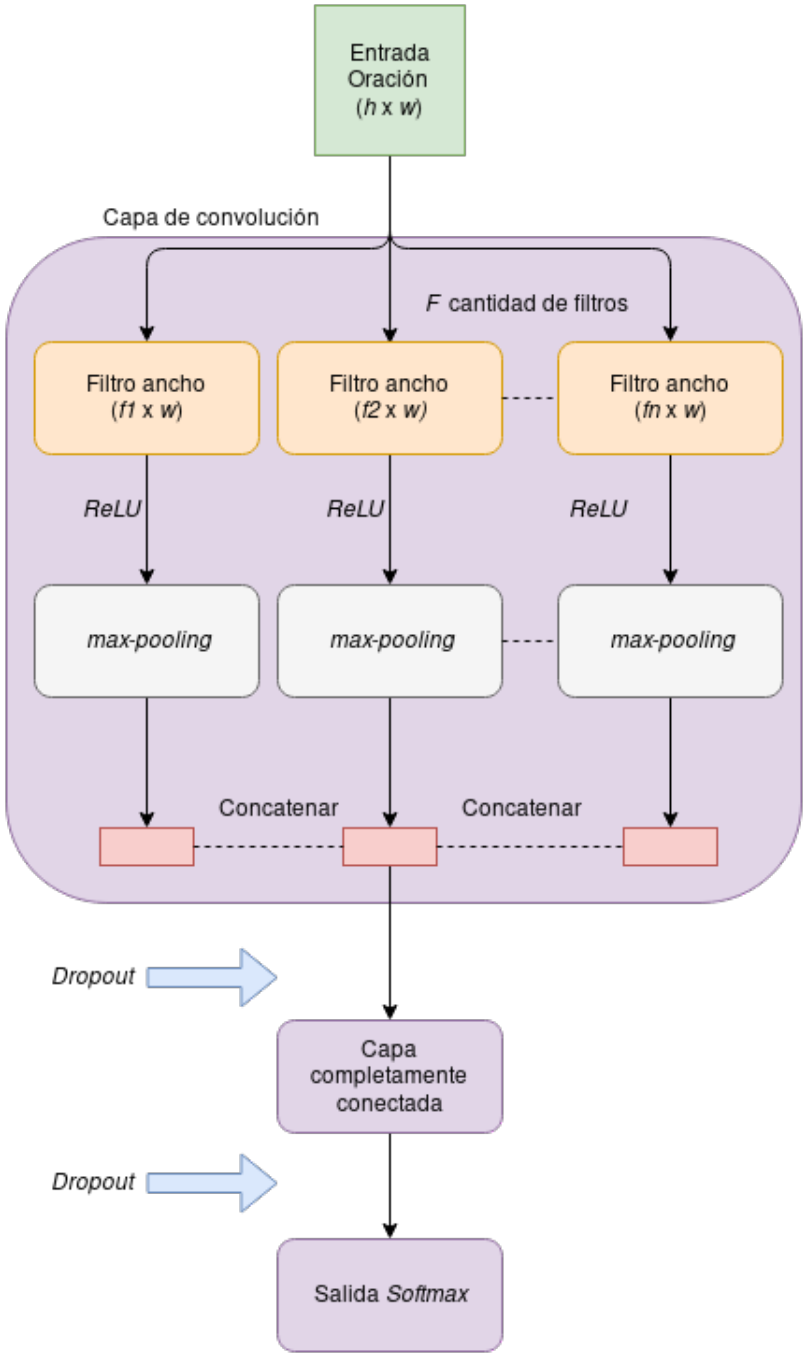

Figura 7: Arquitectura de la red neuronal convolucional desarrollada (Elaboración propia).

$$A^1 = concatenate(A^{11}, A^{12}, A^{13})$$

La operación de multiplicación (.) es reemplazada por la convolución ($*$) y el parámetro de pesos W por filtros f. Se suma el bias b como antes y se realiza la evaluación de la función de activación, en este caso *ReLU*. Posterior a la activación no lineal se realiza la operación

de *pooling*, finalmente todos los resultados son concatenados para conformar el vector de activación de la capa actual. Estas fórmulas se limitan a mostrar un ejemplo con solo tres filtros, pero el mismo principio se aplica para F cantidad de filtros, aunque se deberá cambiar la dimensión de los parámetros en capas posteriores.

En la segunda capa se realiza primero *dropout*, proceso realizado únicamente durante el entrenamiento, sobre las activaciones de la capa anterior.

$$A^1 = dropout(A^1)$$

$$Z^2 = W^2 . A^1 + b^2$$

$$A^2 = relu(Z^2)$$

Las últimas dos capas son clásicas, completamente conectadas, es decir se realiza la multiplicación usual entre matrices y activación no lineal sobre cada elemento del resultado. La tercera capa difiere en la función de activación ya que se utiliza *softmax* para realizar la clasificación.

$$A^2 = dropout(A^2)$$

$$Z^3 = W^3 . A^2 + b^3$$

$$A^3 = softmax(Z^3)$$

Estas fórmulas resumen el procedimiento $propagacion\_hacia\_adelante(...)$ visto en los Algoritmos 2 y 3, por lo que ya es posible implementar la evaluación de la red neuronal, pero no es suficiente para el entrenamiento. Debido a las ventajas que ofrecen las librerías de *Deep learning*, desarrolladas con el auge de la minería de datos, no es necesario implementar las operaciones de convolución, multiplicación o *pooling* por cuenta propia, lo mismo se aplica para el algoritmo de *backpropagation* que es necesario para el procedimiento $propagacion\_hacia\_atras(...)$.

Pero aunque no se necesite implementar la $propagacion\_hacia\_atras(...)$ es aún necesario elegir la función de costo. Debido a los objetivos de este trabajo, la función conocida como **entropía cruzada promedio** es la elegida por tratarse de una tarea de clasificación.

$$J = \frac{1}{m} \sum_{i=0}^{m} D(A_i^3, L_i)$$

Se utilizan todos los m elementos del conjunto de entrenamiento para evaluarla, el uso de

esta función es una de las razones del bajo rendimiento en tiempo del descenso de gradiente y mencionado favoritismo a su extensión el algoritmo *SGD*.

El tensor A_i^3 es la activación de la última capa de la red, es decir las probabilidades de cada etiqueta. L representa los resultados esperados. Ambos A_i^3 y L son comparados con la entropía cruzada denotada por D (distancia).

$$D(s,l) = -\sum_{i=0}^{\#etiquetas} l_i \log(s_i)$$

Los vectores s y l tienen tantos elementos como salidas la red neuronal. Se calcula la distancia entre una salida probabilística y el resultado esperado.

4.5. HERRAMIENTAS E INFRAESTRUCTURA

En años recientes las áreas de Inteligencia Artificial han tenido una influencia importante por su uso comercial, motivo por el cual lenguajes como *Python* o R han ganado popularidad y *frameworks* especializados como *Tensorflow*[2] han aparecido.

El lenguaje de programación *Python* en especial ha ganado gran popularidad en *Machine learning*, ya que son muchas las librerías para minería de datos que han aparecido para esta tecnología. Herramientas en verdad poderosas que ahorran mucho tiempo en el desarrollo de redes profundas y más complejas. El *framework* más popular en *Python* es *Tensorflow*, que también tiene interfaces para otros lenguajes como C y C++.

Tensorflow es un *framework* de código abierto especializado en cálculo numérico mantenido por *Google*[3], es la herramienta más utilizada y popular para *Deep learning* debido a su desarrollo continuo, flexibilidad y extensibilidad. Los conceptos fundamentales detrás de este *framework* pueden deducirse de su nombre "flujo de tensores", que se refiere a la propiedad de administrar todas sus operaciones en un grafo, donde al evaluar un tensor se ejecuta un flujo, dirigido por las operaciones que lo conforman. Esta estructura se conoce como **grafo de computación** y es la base fundamental de *Tensorflow*.

Como se sabe, para entrenar una red neuronal se debe calcular el gradiente de la función de costo, necesaria para el paso de actualización de pesos en el algoritmo de descenso de gradiente o *SGD*. Hace años era necesario que uno mismo implementará el algoritmo de

[2]https://www.tensorflow.org
[3]https://www.google.com

backpropagation para lograr esto, pero hoy en día los *frameworks* como *Tensorflow* se encargan de calcular las derivadas parciales, ahorrando a los programadores la necesidad de implementarlo por cuenta propia. Esta es una de las razones por la que cada vez más gente se une a este tipo de proyectos sin siquiera saber cómo implementar los algoritmos de entrenamiento de una red neuronal, y no tienen que saberlo. Algunos *frameworks* abstraen tanto los detalles que con pocas líneas de código se puede tener corriendo una red neuronal en pocas horas.

Para bien o para mal, *Tensorflow* no es un *framework* minimalista, sino que los detalles internos de la implementación de una red neuronal son expuestos y el usuario es quien debe definir el diseño de la red por cuenta propia. Esto también puede verse como ventaja, ya que a diferencia de otros *frameworks*, permite crear modelos personalizados y más complejos que no serían posibles de lograr con otras librerías. En sí el diseño de la red neuronal ya ha sido presentado, adicionalmente en esta sección se explicarán detalles de su implementación.

Para poder entrenar la redes neuronales que probarían la hipótesis del trabajo, se desarrolló una pequeña infraestructura que permite probar diferentes arquitecturas y modificar los hiperparámetros de manera sencilla. Esta infraestructura fue desarrollada encima de *Tensorflow* con el lenguaje de programación *Python*, en la Figura 8 se muestra un diagrama de clases que representa el diseño interno de este programa.

La clase **SACNN** es la principal, está compuesta por un objeto arquitectura que luego se puede entrenar, guardar, restaurar, evaluar y utilizar para realizar clasificaciones. La arquitectura de una red neuronal está abstraída en la clase **Arch**. En el diagrama de clases solo se ven dos especializaciones las cuales son Kim y un diseño más avanzado, que se denominará "Evolucionada", haciendo alusión a que su diseño creció durante la experimentación. Esta disposición se conoce como el patrón de diseño **estrategia** y permite extender las arquitecturas disponibles con solo añadir una clase.

El patrón estrategia también se puede ver en la interfaz **TrainIterator** que básicamente indica el algoritmo de aprendizaje que se utilizará. Se tiene implementado el algoritmo *SGD* dentro las dos opciones disponibles, la diferencia radica en que la clase **SimpleItarator** no tiene modificaciones al algoritmo pero **EarlyStopIterator** tiene la parada temprana implementada.

Las clases de la derecha inferior **KimSACNNBuilder** y **EvolvedSCANNBuilder** son constructores especializados que pueden devolver una objeto **SACNN** de una arquitectura específica a través de una interfaz más sencilla. El objeto SACNN retornado puede ser nuevo, entrenado o restaurado, esto es posible debido a las diferentes abstracciones que el constructor

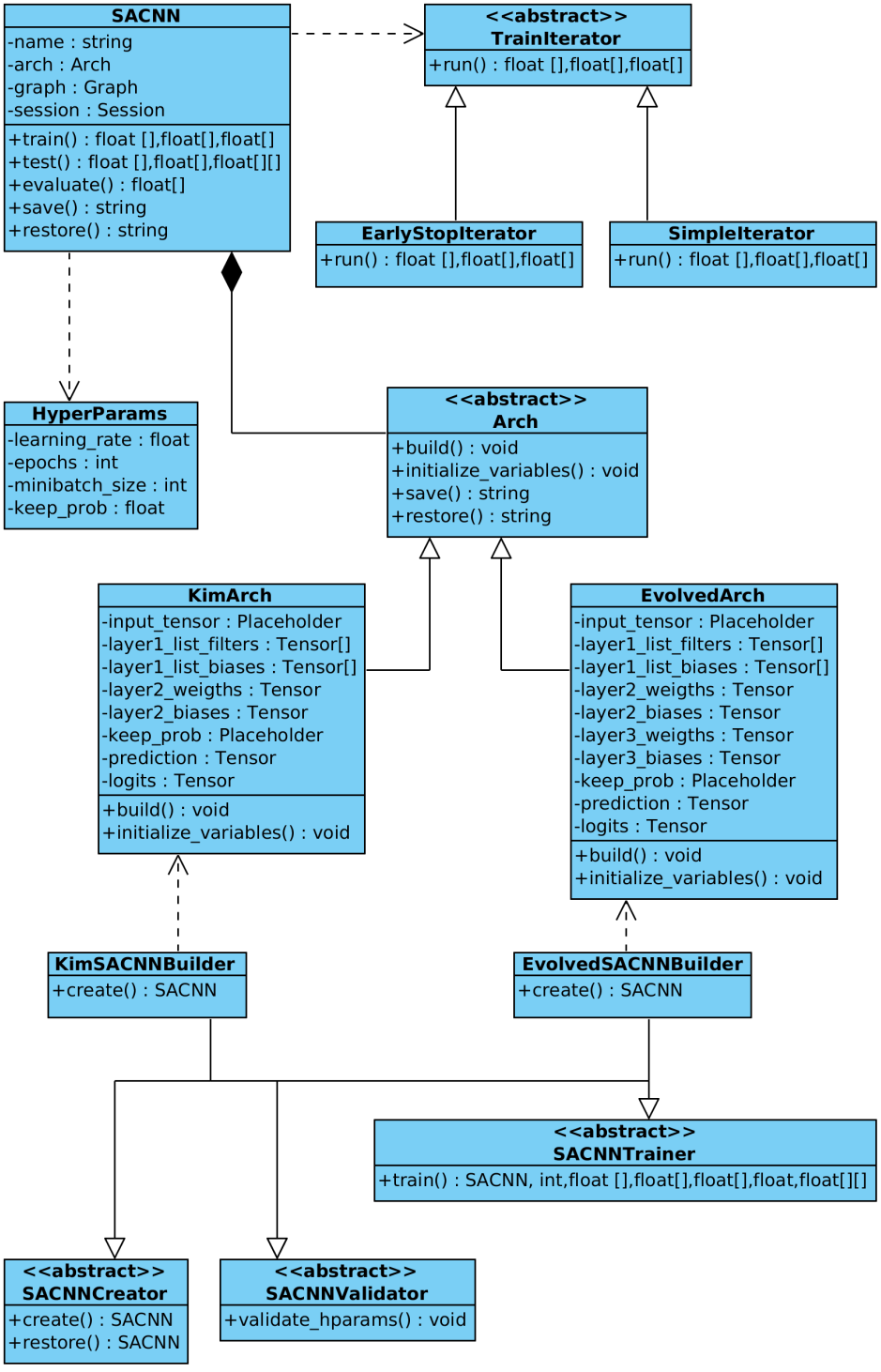

Figura 8: Diagrama de clases de la infraestructura creada para entrenar la red neuronal con *Tensorflow* (Elaboración propia).

hereda de sus padres. En un lenguaje como *Python* donde la herencia múltiple es posible se puede realizar este patrón el cual es conocido como **mixin**.

4.6. MARCO DE TRABAJO

El proceso de desarrollo de un modelo de *Deep learning* que resuelva correctamente el problema requiere mucho tiempo para realizar las pruebas y experimentación. El progreso de este tipo de proyectos no cuenta con metodologías ampliamente aceptadas por lo que no se puede mencionar alguna, pero no se ha realizado esta investigación a ciegas. A continuación se mostrará el marco de trabajo referencial seguido, el cual está inspirado en los concejos de Goodfellow y cols. (2016).

El flujo de trabajo se resume en la Figura 9, consta de tres etapas principales en el camino normal antes de realizar la evaluación de desempeño. La primera etapa **recolección de datos** se refiere a la obtención, selección y control de calidad de los datos, es decir los conjuntos de entrenamiento, validación y prueba. La etapa de recolección ya se tocó completamente en el Capítulo 3 de este documento.

La etapa de **diseño** comprende principalmente el desarrollo de la arquitectura de la red neuronal. Esta es una etapa importante ya que se deciden muchos de los hiperparámetros del modelo. Lo normal es utilizar una arquitectura que funciona para un problema similar al que se quiere resolver, si esta no funciona se elige otra o se ajusta según sea necesario. En esta etapa también se elige el algoritmo de entrenamiento ya que esta es una parte importante del diseño.

Aún a pesar que el algoritmo de entrenamiento se elige en la etapa anterior, la tercera etapa del marco de trabajo se ha nombrado **entrenamiento**. Esta etapa indica la terminación de la implementación y ejecución de la red neuronal, es decir la realización del entrenamiento. El sistema es desplegado y ejecutado recibiendo como argumentos los conjuntos de datos, la implementación de la arquitectura y los hiperparámetros necesarios. Dependiendo del ambiente o los requerimientos, se puede entrenar varias redes dentro un procedimiento automatizado y así realizar pruebas para encontrar la configuración que mejor rendimiento tenga.

Luego de entrenar la red es necesario realizar la **evaluación** de la misma para saber si su desempeño es lo suficiente bueno. El umbral de decisión varía para cada caso. Se determina la efectividad de la red con dos preguntas. La pregunta sobre parcialidad se refiere a que tan bien se predice sobre el conjunto de entrenamiento, si ni siquiera capaz de generalizar el

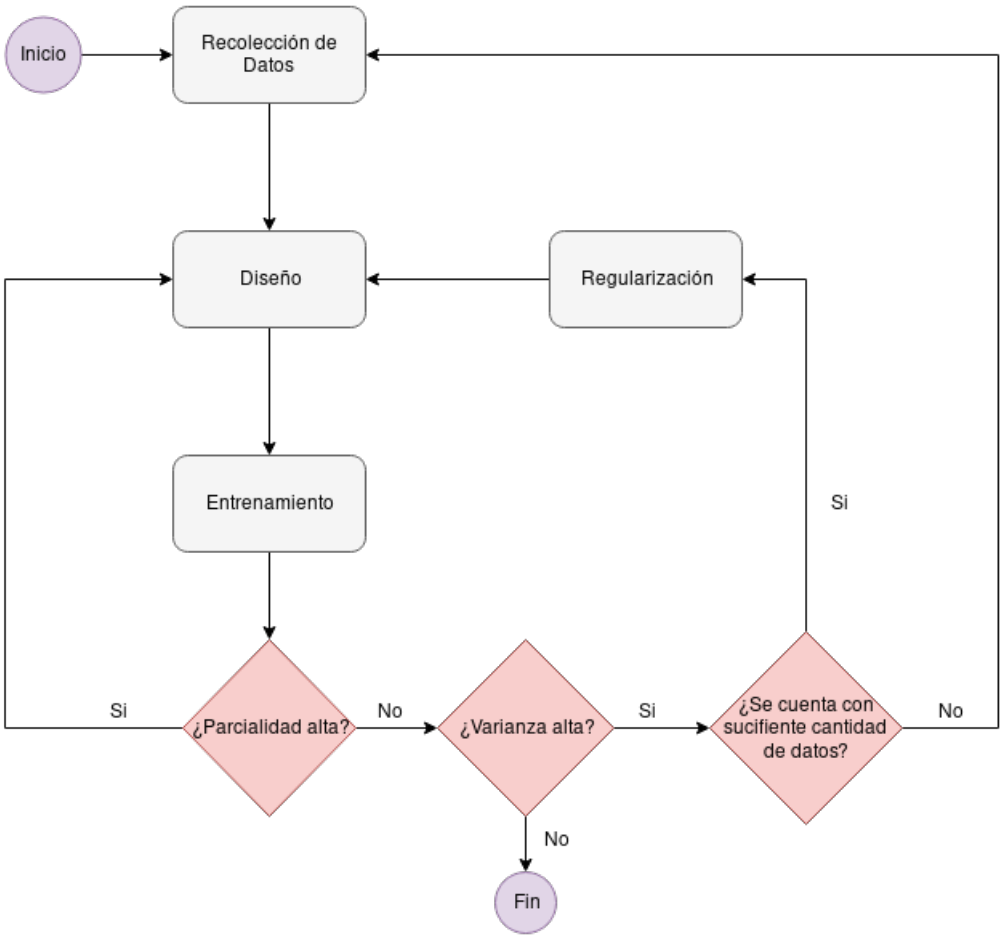

Figura 9: Marco referencial de trabajo para el desarrollo de la red (Elaboración propia).

conjunto de entrenamiento el diseño es fallido por lo que es necesario reformular el plan. Si la parcialidad no es alta entonces se pregunta si la varianza es alta, ya se ha mencionado este término antes[4]. Si la varianza no es alta se dice que los parámetros ya son los óptimos y el trabajo ha terminado.

Si la varianza es alta se puede abordar el asunto de dos formas diferentes para lo que es necesario conocer el estado de los componentes del modelo. Si la cantidad de datos es pequeña lo más probable es que un diseño complejo está aprendiendo demasiado rápido el conjunto de entrenamiento, en ese caso encontrar más datos puede ayudar. Otra opción es reducir la complejidad del diseño utilizando una arquitectura menos profunda. Si se cree que se tienen suficientes datos o éstos son limitados se puede probar con una técnica de regularización, para ello es necesario decidir dónde de aplicará por lo que se vuelve a la etapa diseño.

[4]Se refiere al desempeño sobre el conjunto de validación que permite observar si existe sobre entrenamiento.

Se puede notar que este marco de trabajo está a favor de un flujo iterativo más que uno cascada, y es que se recomienda empezar con el desarrollo en etapas tempranas de la planeación ya que es muy probable que necesidades impredecibles surjan con el tiempo. Es mejor empezar con un conjunto de datos pequeño y tratar de mejorar el desempeño desde el inicio que tardar seis meses recolectando datos y seis más intentando encontrar el error.

Este trabajo empezó con una cantidad de datos muy limitada, esto es menos de 500 ejemplos. En las primeras iteraciones el desempeño no era bueno y el diseño se sobre entrenaba demasiado rápido, con el tiempo se recolectaron más datos y el rendimiento mejoró poco a poco. También a ayudó realizar nuevos ajustes sobre los hiperparámetros del diseño original.

Se notó que el uso de filtros de mayor altitud es un elemento que influye fuertemente en el sobre entrenamiento de la red. También se probó con diferente número y tamaño de filtros. Lo más interesante fue saber que los filtros cuadrados logran un rendimiento muy bajo, a diferencia de los anchos, por lo que no son los mejores para este tipo de tareas.

Una red más profunda incrementa la complejidad tanto que el desempeño sobre datos nuevos es muy pobre, es por eso que las instancias que mejor resultados dieron no son muy diferentes del diseño original. Se descartó realizar *pooling* con ventanas más pequeñas debido al costo computacional, por ello es importante tener una gran cantidad de filtros, para evitar perder mucha información.

CAPÍTULO 5 APLICACIÓN Y EXPERIMENTACIÓN

Este capítulo tiene dos objetivos, mostrar una aplicación que hace uso de redes neuronales convolucionales con ayuda de la infraestructura presentada en el capítulo anterior, y presentar los resultados de las pruebas realizadas durante la etapa de experimentación. Se mostrarán las diferentes configuraciones de la arquitectura desarrollada, exhibiendo la precisión obtenida sobre un conjunto de datos de prueba. Al final, se tiene un apartado destinado a comparar el desempeño del modelo desarrollado en este trabajo contra técnicas de *Machine learning* ampliamente probadas en problemas de análisis de sentimientos.

5.1. CLASIFICADOR DE OPINIONES POR LOTE

Esta sección mostrará un sistema web que hace uso de la infraestructura mostrada en el Capítulo 4 para restaurar y entrenar redes neuronales con el objetivo de realizar clasificaciones. Al tratarse de un sistema web este consiste de una arquitectura cliente-servidor, la infraestructura junto a otras clases que levantan el servicio conforman el lado del servidor. El cliente es relativamente sencillo, ya que solo envía comentarios y recibe los resultados.

El proceso de clasificación de un comentario se resume en la Figura 10. Un comentario, como el que se puede ver en la imagen, es procesado para que pueda ser utilizado como entrada de una red neuronal convolucional. Se debe usar el mismo *word embedding* con el que se contó para realizar el preprocesamiento del conjunto de entrenamiento de dicha red. El comentario se convierte en una matriz $(n \times d)$ donde n es la cantidad de palabras máxima que se usarán y d es la dimensión del *word embedding*.

El sistema utiliza la infraestructura la cual se encarga de la restauración o entrenamiento de nuevas redes. Cuando es momento de que clasificar un comentario ya procesado, la infraestructura funciona como una capa de abstracción entre los oyentes de los eventos

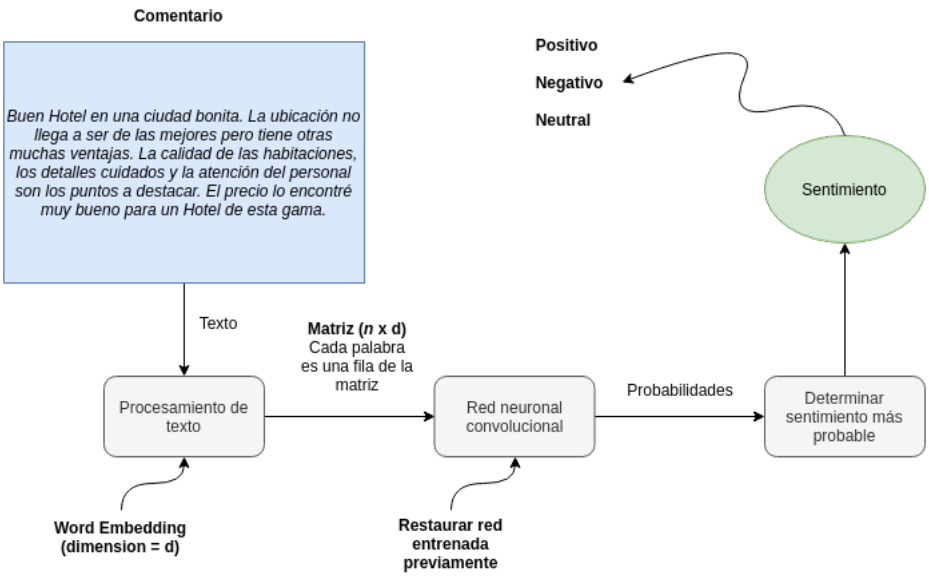

Figura 10: Proceso de clasificación de comentarios (Elaboración propia).

HTTP[1] y la red neuronal misma. En la Figura 11 se muestra un diagrama que indica la disposición y comunicación entre los componentes del sistema. La petición que realiza el cliente especifica qué red se utilizará para clasificar el comentario, así es posible obtener resultados de diferentes redes en cada solicitud.

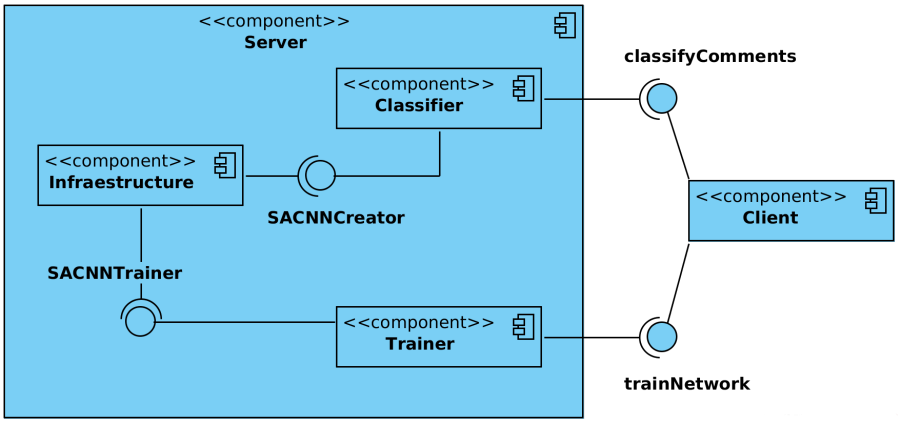

Figura 11: Composición del clasificador de opiniones por lote (Elaboración propia).

La red neuronal convolucional devuelve un vector de probabilidades de tamaño igual al número de etiquetas, pero estos valores no son devueltos al cliente. Antes de terminar de procesar una solicitud se determina el sentimiento basando en las probabilidades retornadas por la red, y se envía este valor.

[1]Protocolo de transferencia de hipertexto.

Se dice que es un clasificador por lotes ya que el usuario puede ingresar a través de la interfaz múltiples comentarios y esperar que los resultados de todos ellos lleguen al mismo tiempo. Esto en realidad no es sorprendente ya que, por cómo está implementada la red neuronal (tensores conectados por operaciones) la clasificación por lotes es su funcionamiento por defecto, de hecho para el caso de clasificar un solo comentario se lo debe manejar como lote de un solo elemento.

5.2. PRUEBAS

La experimentación de este trabajo siguió un proceso iterativo como se explica en el Capítulo 4, por lo que no sería eficiente mostrar al detalle este desarrollo. Se explicarán las pruebas realizadas en tres etapas, las cuales resumen los eventos más significativos de la evolución del modelo.

Consiste en entrenar cierta cantidad de redes neuronales con diferentes configuraciones, es decir, variaciones en los hiperparámetros de la arquitectura general (ver Capítulo 4) con tal de conocer cuál funciona mejor. Con esta información uno puede decir que se alcanzaron los resultados esperados y terminar la experimentación, o realizar más pruebas buscando mejorar aún más el desempeño.

La **primera etapa** consiste en evaluar una arquitectura igual a Kim (2014), es así porque son los inicios la experimentación. Esta arquitectura consiste de una capa de convolución con filtros anchos los cuales pueden tener un alto distinto en la misma capa, seguidamente tiene una capa *softmax* la cual realiza la clasificación. Es en realidad sencilla en términos de profundidad pero compleja en lo que a operaciones sobre tensores se refiere. Las pruebas se centraron en dos hiperparámetros, la cantidad o **número de filtros** y el **alto** de ellos. Se nombrará este diseño como **Kim** para diferenciarlo de los que se presentarán más adelante.

En esta etapa inicial, el diseño fue aún más sencillo de lo que presenta Kim (2014) porque no se utilizaron distintos altos de filtro al mismo tiempo. Adicionalmente, todas las pruebas se realizaron dos veces, una para cada algoritmo de entrenamiento presentado en este documento (*SGD* y *SGD* con parada temprana).

Los resultados se muestran en la Figura 12, el cuadro de la izquierda corresponde a las pruebas con *SGD* normal y a la derecha con parada temprana. De la primera gráfica se puede notar que un número de filtros menor tiene mejores resultados cuando tienen alto pequeño, por otro lado mientras se incrementa la cantidad de filtros el desempeño también lo hace sin importar mucho el alto del filtro. En la segunda gráfica se ven diferencias en cuanto a la

precisión con menos cantidad de filtros y es que hay una pequeña mejora.

Para la **segunda etapa** lo que se hizo fue intentar con una arquitectura más profunda. Se añade una capa de neuronas completamente conectadas (FC)[2] antes de la capa *softmax* tomando inspiración de Hughes y cols. (2017), los detalles de esta composición se explicaron en el Capítulo 4. A diferencia del anterior se conocerá a esta arquitectura como **Kim+1FC**. Las pruebas consisten en variar el **número de unidades** en la capa recientemente añadida, utilizando los filtros que mejores resultados dieron en la etapa anterior. Al igual que antes se realizarón las pruebas con ambos algoritmos de entrenamiento.

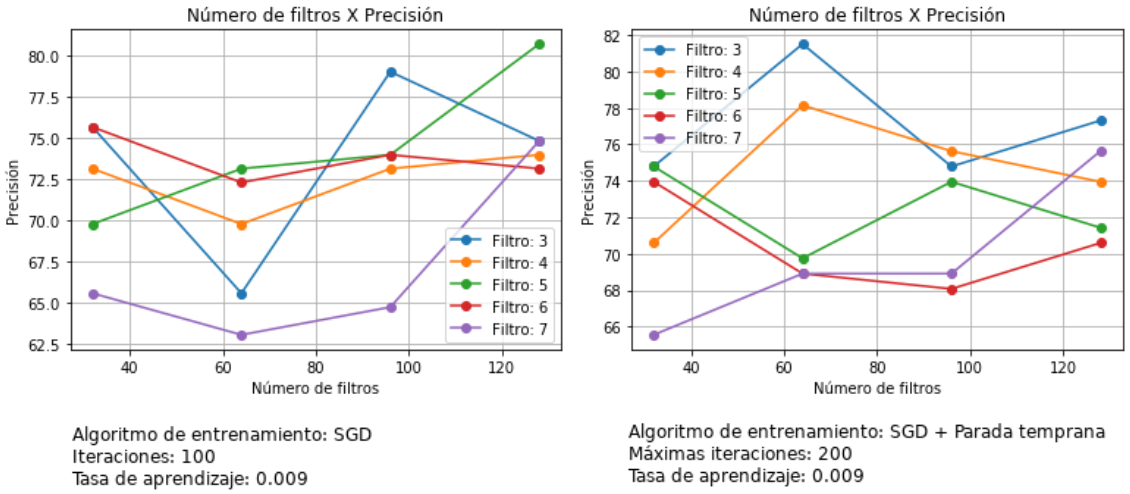

Figura 12: Pruebas de la etapa 1, número de filtros contra precisión (Elaboración propia).

La curva izquierda en la Figura 13 muestra que con un número fijo de iteraciones los desempeños lucen similares y constantes a pesar del número de unidades. Por otro lado, el algoritmo de parada temprana obtiene mejores resultados con bastante cantidad de filtros. En esta etapa se decidió aumentar en 100 el número de iteraciones, ya que se noto que el modelo no entrenaba lo suficiente.

En la **tercera etapa** la última de esta investigación, se realizó un análisis enfocado en tener filtros con **diferentes altos** en una misma capa de convolución. A esta forma de hacer funcionar la primera capa se llamará convoluciones en paralelo (PC)[3]. Las configuraciones de las etapas pasadas **Kim** y **Kim+1FC** fueron utilizadas como base para estas pruebas. Se conformaron dos diseños a evaluar, **Kim+PC** y **Kim+PC+1FC** y al igual que antes las configuraciones que mejores resultados dieron se utilizaron como punto de partida.

[2]Por sus siglas en inglés, *Fully Connected*.
[3]Por sus siglas en inglés, *Parallel Convolutions*.

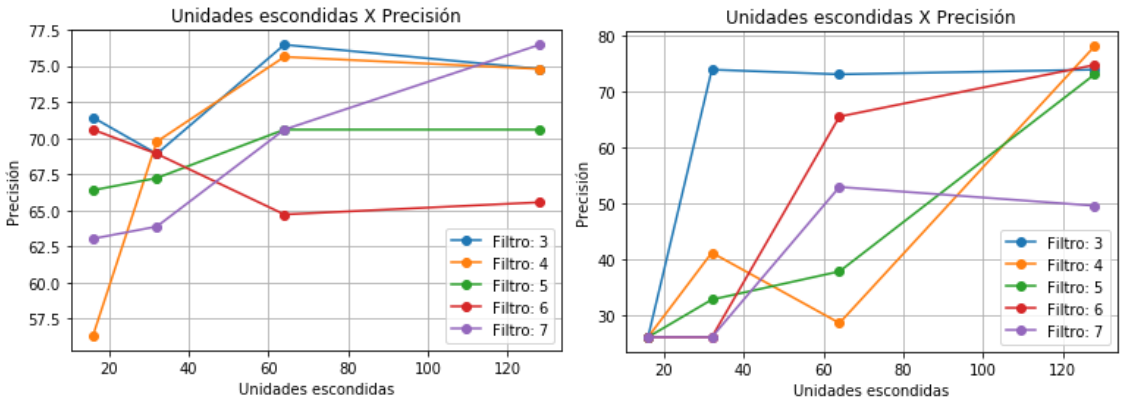

Figura 13: Pruebas de la etapa 2, número de filtros contra precisión (Elaboración propia).

Las gráficas están en función al número de convoluciones en paralelo (ver Figura 14). Se puede ver que una arquitectura menos compleja como **Kim+PC** (sin la capa escondida extra) tiene en general mejor desempeño, aunque utilizando parada temprana la precisión desciende un poco. Al parecer **Kim+PC+1FC** mejora su rendimiento con el algoritmo de entrenamiento modificado, esto es de esperarse ya que al tratarse de un diseño más complejo aprende los ejemplos a un ritmo diferente (se intuye que necesita menos iteraciones para generalizar).

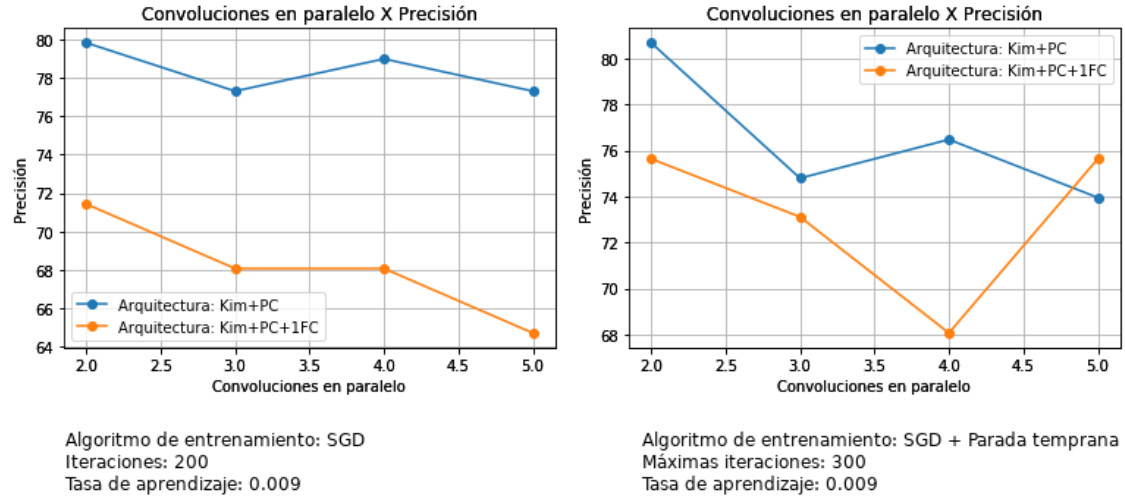

Figura 14: Pruebas de la etapa 3, convoluciones en paralelo contra precisión (Elaboración propia).

Como se pudo observar hasta ahora las configuraciones no son nada espectaculares, esto es debido a que los datos de entrenamiento (el *corpus*) es limitado. Las pruebas fueron ajustadas a la cantidad de ejemplos con la que se cuenta.

Hiperparámetro		**Hiperparámetro**	
Tasa de aprendizaje (α)	0.009	Tamaño de oración (h)	100
Tamaño de minilote	32	Dimensión de *word embedding*(d)	300
Tasa de *dropout*	0.5	Paso (s)	1
Paciencia (P)	5	Relleno (p)	0
Función e activación (g)	*ReLU*		

Tabla 5: Hiperparámetros de la red neuronal convolucional que permanecieron constantes entre las pruebas.

En estas pruebas se mostró cómo el diseño base evolucionó modificando su forma o valores de hiperparámetros, aunque solo algunos fueron objeto de análisis ya que es muy costoso realizar las mismas pruebas para cada configuración posible. Los hiperparámetros que se muestran en la Tabla 5 son aquellos que no se modificaron entre pruebas, es decir fueron constantes. Se podría añadir el número de iteraciones utilizadas para entrenar un modelo (N), pero ya que existen diferencias entre la primera etapa y el resto se decidió no listarla.

Arquitectura	**Alto de filtro**	**Número de filtros**	**Unidades escondidas**	**Algoritmo de entrenamiento**	**Iteraciones / Máximas iteraciones**
Kim	5	128	No aplica	*SGD*	100
Kim+1FC (1)	3	96	64	*SGD*	200
Kim+1FC (2)	7	128	128	*SGD*	200
Kim+PC	4/6	128/32	No aplica	*SGD*	200
Kim+PC+1FC	5/7	128/128	128	*SGD*	200
Kim	3	64	No aplica	Parada temprana	200
Kim+1FC	4	64	128	Parada temprana	300
Kim+PC	3/5	96/128	No aplica	Parada temprana	300
Kim+PC+1FC (1)	3/5	64/32	128	Parada temprana	300
Kim+PC+1FC (2)	3/4/5/ 6/7	64/64/32/ 32/128	64	Parada temprana	300

Tabla 6: Los hiperparámetros variables que mejor desempeño mostraron en las pruebas.

Por otro lado, en la Tabla 6 se recopilaron los hiperparámetros que tuvieron el mejor desempeño de todas las etapas (la precisión de estos se discutirá en la siguiente sección del capítulo). Debido al pequeño espacio de decisión y tamaño del conjunto de pruebas, algunas arquitecturas con diferente configuración comparten la misma precisión. Esto no debería

sorprender si se ve con cuidado la tabla comparativa de la siguiente sección, sucede lo mismo con otros métodos. Para diferenciar conjuntos de hiperparámetros de una misma arquitectura con igual precisión se utilizó un índice, el cual se puede ver aplicado en la primera columna.

Por simplicidad se utilizará el nombre **Parada temprana** para hacer referencia al algoritmo de entrenamiento *SGD* modificado. Se puede ver su uso en la recientemente presentada Tabla 6 y en la siguiente sección, cuando se comparen las arquitecturas desarrolladas contra métodos ampliamente aceptados para problemas de clasificación.

5.3. COMPARACIÓN

Es momento de presentar el desempeño de las arquitecturas presentadas con números. El criterio que se utiliza para medir este rendimiento es la **precisión**, la cual consiste en evaluar un conjunto de ejemplos etiquetados y compararlos contra los verdaderos resultados. Es la cantidad de veces que la red evaluó correctamente, dividido entre el número total de ejemplos multiplicado por 100.

$$precision = \frac{\#aciertos}{\#total} * 100$$

En esta sección se comparan los diseños de las redes neuronales convolucionales presentadas en la sección anterior contra métodos aceptados, con tal de probar la hipótesis de este trabajo. En la actualidad los métodos de clasificación que representan el estado del arte para análisis de sentimientos son *SVM*, regresión logística y redes neuronales superficiales.

La Tabla 7 muestra finalmente esta comparación, se puede ver que los métodos que mejor resultados presentan son **Kim** con parada temprana y **Regresión logística**. Mencionar que para obtener estos resultados se tuvo que entrenar todos métodos con el *corpus* conformado para este trabajo, porque de otra forma no tendría sentido realizar esta comparación (los dominios de datos deben ser los mismos).

De esta tabla comparativa se puede deducir también que el algoritmo de entrenamiento *SGD* modificado (con parada temprana), mejora significativamente la precisión de la red. Cómo se mencionó en su momento, la parada temprana evita que el modelo se sobre entrene deteniendo el algoritmo en el momento que el costo sobre el conjunto de validación empieza a subir. En la Figura 15 se muestran las curvas de aprendizaje de dos diseños. Ahí se puede ver que el entrenamiento se detuvo antes de alcanzar el máximo de iteraciones permitidas ya que

Método	Precisión	
/	*SGD*	Parada temprana
Kim	80.67	**81.51**
Kim+1FC	76.47	78.15
Kim+PC	79.83	80.67
Kim+PC+1FC	71.43	75.63
SVM	78.15	
Regresión logistica	**81.51**	
Red neuronal superficial	75.63	

Tabla 7: Comparación de desempeño para análisis de sentimientos con el *corpus* conformado.

el error empezaba a subir. En el caso de **Kim**, el diseño que tiene mejor rendimiento, su curva de aprendizaje generaliza suavemente el conjunto de datos, a diferencia de **Kim+1F** el cual es un diseño más complejo[4] por lo que aprende demasiado bien en las primeras iteraciones tanto que llega a sobre entrenar un poco.

Debido al algoritmo *SGD* las curvas de aprendizaje tienen esa forma peculiar parecida a un garabato. La red se entrena con mini lotes generados aleatoriamente en cada iteración, por lo que el aprendizaje es un poco alborotado con altos y bajos como se puede ver en las gráficas. A pesar de su apariencia el algoritmo funciona bastante bien.

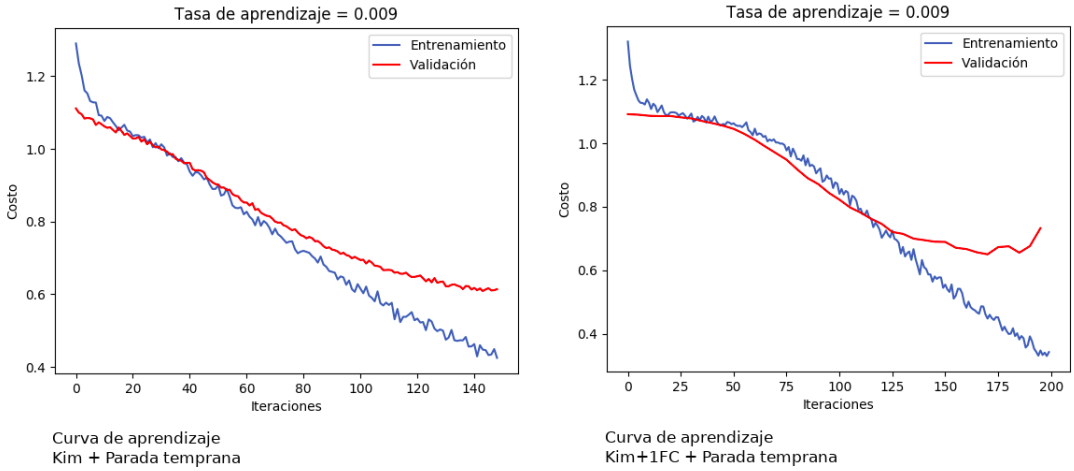

Figura 15: Comparación de las curvas de aprendizaje de las redes entrenadas con parada temprana (Elaboración propia).

Con esto se demuestra la efectividad de las redes neuronales convolucionales para realizar la tarea de análisis de sentimiento. Se puede decir que los resultados son muy similares al

[4]Que aprende los datos con funciones de más alto orden, lo que resulta una ventaja para datos con alta dimensionalidad.

estado del arte como menciona Kim (2014) en su trabajo.

CAPÍTULO 6 CONCLUSIONES Y RECOMENDACIONES

A lo largo de esta investigación se ha experimentado con diferentes diseños de red neuronal convolucional con el fin de responder a la pregunta: ¿Cuáles son los parámetros de las redes neuronales convolucionales que influyen en la exactitud? Los elementos que se identificaron en la planeación fueron el diseño de la red y la calidad de los datos de entrenamiento, a continuación se exponen las conclusiones al respecto:

- Empezando por el diseño o arquitectura de la red, al tratarse de una red neuronal convolucional lo más determinante fueron sin duda los filtros anchos. Un filtro es ancho cuando comparte el tamaño de uno de sus lados con el mapa de características. El lado compartido normalmente corresponde al lado dimensional del *word embedding*, es decir, el ancho de un filtro abarca toda la palabra embebida en el mapa de características. El alto, el cual es un hiperparámetro, indica cuántas palabras se procesan a la vez en la convolución. Utilizar *dropout* para regularización y *softmax* para clasificación en la última capa son otros factores importantes de la arquitectura. La regularización evita que la red aprenda en exceso los datos de entrenamiento ayudando a tener una mejor generalización. La capa *softmax* es especialista en clasificación de datos multiclase, así que es idóneo para análisis de sentimientos.

  Las distintas configuraciones de hiperparámetros probadas, también son concernientes al diseño. Las arquitecturas de redes neuronales convolucionales evaluadas consisten en general de tres capas, pero en algunas configuraciones la segunda es omitida. La primera capa es de convolución, los hiperparámetros concernientes son el alto de los filtros y la cantidad de ellos. La segunda es una capa de neuronas completamente conectada, añadida durante la experimentación con la intención de mejorar los resultados, el número de unidades es un atributo variable de esta capa. La última capa *softmax* no cambió sus valores entre pruebas. Los hiperparámetros probados durante la experimentación fueron principalmente el alto de los filtros, la cantidad de ellos y el

número de unidades escondidas de la segunda capa. Adicionalmente, todas las pruebas se realizaron dos veces para experimentar con el algoritmo *SGD* y parada temprana.

Lo más concluyente que se puede obtener de los experimentos con diferentes configuraciones de redes neuronales convolucionales, es que el algoritmo de parada temprana tendría un impacto positivo en lograr una generalización más apropiada.

Por otro lado, los resultados de las pruebas sobre los filtros o unidades escondidas no muestran un patrón claro que permita asegurar un camino correcto a seguir. Puede deberse a que las pruebas cubren un rango muy bajo de los factores determinantes para la precisión, por lo que experimentos con otros hiperparámetros o técnicas de regularización podrían ayudar a mejorar la precisión. Sin embargo, es probable que el origen de este problema sea el limitado tamaño del *corpus*.

- En cuanto a la calidad de datos de aprendizaje se refiere, se hizo un trabajo minucioso en la recolección y control de calidad de un conjunto de opiniones (*corpus*).

  Durante la recolección entraron en juego un grupo de criterios para determinar un dominio de datos pertenecientes al turismo en Bolivia. Los criterios de selección elegidos fueron idioma español, localidad Bolivia, opiniones escritas en la web más recientes de preferencia y que el contenido del texto trate sobre atractivos turísticos sin importar la variedad de palabras con las que esté escrito.

  La calidad de los datos recolectados fue medida a través del cumplimiento de las propiedades: representatividad, balance, longitud y homogeneidad. Estas propiedades se midieron de una forma interpretativa, ya que no se contó con el tiempo de implementar técnicas que calculen el cumplimiento con números.

  - Se cumplió con la representatividad al alcanzar una cantidad de ejemplos mayor al objetivo inicial, se conformó un *corpus* de 1200 ejemplos. Son tres los sentimientos que se manejaron en este trabajo: positivo, negativo y neutral, cada uno de ellos contando con 400 ejemplos pertenecientes al *corpus* total.
  - Se cumplió la propiedad de balance al tener grupos de datos con la misma etiqueta de igual tamaño.
  - La homogeneidad también se cumplió, porque el *corpus* pertenece a un dominio controlado por los criterios de selección.
  - La propiedad dio algunos problemas para poder cumplirla y es que indica que los ejemplos del *corpus* deberían tener tamaño predecible (como son oraciones se refiere a la cantidad de palabras), sin embargo, las opiniones que se puede

encontrar en la web son variables. Debido a que no es posible predecir el tamaño de una oración la solución es tener un límite de palabras permitidas con la inevitable pérdida de información como consecuencia. Cómo el objetivo es que el algoritmo de *Machine learning* no pierda información por tener textos de tamaño variable, la solución es controlar la pérdida antes del entrenamiento así el impacto no es notorio. Adicionalmente el uso de *word embedding* agrega más valor al *corpus* gracias a la transferencia de aprendizaje.

En esta investigación se ha experimentado con diferentes diseños de red neuronal convolucional con el propósito de automatizar la clasificación de opiniones turísticas de Bolivia. El modelo con la precisión más alta consiste de la arquitectura denominada **Kim**, entrenado con el algoritmo *SGD* y la parada temprana. La precisión resultante es **81.51**, alcanzando al desempeño de otros métodos como por ejemplo regresión logística.

Se puede notar que **Kim** es la arquitectura menos compleja presentada en este trabajo y resulta ser la que tiene mejor precisión. De esto se deduce que a pesar de que se cumplieron con las propiedades que califican el *corpus*, el problema puede extenderse aún más. Esto no quiere que los resultados obtenidos sean malos, sino que el *corpus* es sencillo (3 etiquetas y 1200 ejemplos) por lo que es de esperar que un diseño de red neuronal poco profundo lo generalice correctamente.

6.1. TRABAJOS FUTUROS

Una de las primeras cosas que se podría probar a partir de esta investigación es incrementar el tamaño del *corpus*. Aunque no se haya mencionado hasta ahora, se probó a extender el conjunto de datos a 5 etiquetas, obteniendo precisiones cercanas al 50 % de acierto con las redes neuronales desarrolladas. Se tiene la noción de que el rendimiento sobre más etiquetas tendría mejoras notables con un *corpus* más grande.

Si se aumentan las etiquetas es probable que **Kim** con las configuraciones desarrolladas en este trabajo, tenga baja precisión en un principio. Habría que aumentar el número de iteraciones lo que implicaría un incremento en el tiempo de entrenamiento. Ahora mismo con un conjunto de entrenamiento de tamaño 954 (80 % de 1200) se tarda más o menos 30 minutos en entrenar una red con 200 iteraciones. Estos problemas podrían solucionarse probando diseños más complejos como **Kim+1F**. Una arquitectura más profunda haría que el incremento de tiempo sea menor, ya que aprendería conjuntos más grandes con casi el mismo número de iteraciones. Para entrenar nuevas arquitecturas sería posible reutilizar la

infraestructura desarrollada.

Si bien, el trabajo consistió en estudiar el desempeño de las redes neuronales convolucionales para análisis de sentimientos de opiniones turísticas, en realidad estas redes podrían ser usadas para cualquier tarea de clasificación de procesamiento de lenguaje natural. En este documento se explicó en detalle el desarrollo de un clasificador de texto usando redes neuronales convolucionales, si se quisiera usar este conocimiento para otro tipo de tareas, cabe mencionar que el uso de *word embeddings* es opcional. Uno podría simplemente convertir las palabras a vectores *1-hot*[1] y usarlos en redes de convolución, aunque su efectividad no está garantizada.

Las redes convolucionales, usadas en este trabajo para clasificación de textos, son normalmente empleadas para clasificación de imágenes. Su amplia aceptación se debe a que proveen de una interpretación que tiene mucho sentido: En la capa de convolución, un filtro empieza aprendiendo bordes, luego formas más complejas como cuadrados, triángulos para luego entender lo que son objetos o rostros. Esta intuición no puede ser aplicada a procesamiento de lenguaje natural, por lo que el uso de convoluciones en otro tipo de tareas que no sean reconocimiento de imágenes siempre se ve con duda. Por ello se podría intentar utilizar otro tipo de redes neuronales más apropiadas para textos, estos son los modelos secuenciales. Se llama secuencial porque evalúa los datos de entrada en orden. Son especialistas en evaluar información encadenada, por ejemplo un texto es secuencial ya que tiene un orden de lectura (izquierda a derecha, en este lado del mundo). Es por eso que podría ser interesante probar el *corpus* con una **red neuronal recurrente** el cual es un modelo secuencial.

Otra ventaja que el uso de un modelo secuencial podría traer, es evitar perder información por la necesidad de utilizar matrices de tamaño fijo para alimentar la red neuronal. Un modelo secuencial se puede adaptar a datos de tamaño indeterminado, lo que convertirían en triviales los problemas encontrados por recolectar oraciones de tamaño variable.

Las aplicaciones del análisis de sentimientos, puede ser variada, por ejemplo la aplicación vista anteriormente: el servicio web que calcula el sentimiento de una opinión enviada en un formulario y posteriormente el servidor se encarga de clasificar el texto para devolver el resultado. En el mundo es posible encontrar más aplicaciones de este tipo, como por ejemplo el nuevo proyecto de *Google MlKit*[2].

El desarrollo de esta investigación fue guiado por un marco de trabajo basado en consejos

[1] Es un vector de palabra es conformado a partir de un vocabulario. Cada posición del vector indica una palabra del vocabulario, entonces se pone un 1 en la posición de la palabra y el resto se llena con ceros.

[2] https://developers.google.com/ml-kit

y experiencia propia. Este también es un aporte que podría ser útil para trabajos futuros relacionados.

BIBLIOGRAFÍA

Cardellino, C. (2016, March). *Spanish billion words corpus and embeddings.* Descargado de http://crscardellino.me/SBWCE/ (Accedido 2018-04-12)

Christopher, B. M. (2006). *Pattern recognition and machine learning* (J. Michael, K. Jon, y S. Bernhard, Eds.). New York, NY, USA: Springer.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., y Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, *12*(Aug), 2493–2537.

Deshpande, M. (2018). *Machine learning for human beings*. Zenva Pty Ltd. Descargado de https://pythonmachinelearning.pro/wp-content/uploads/2018/01/Machine-Learning-for-Human-Beings-2.pdf (Accedido 2018-02-01)

Dos Santos, C. N., y Gatti, M. (2014). Deep convolutional neural networks for sentiment analysis of short texts. En *Coling* (pp. 69–78).

Dumoulin, V., y Visin, F. (2016). A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*.

Evans, D. (2007). Corpus building and investigation for the humanities. (Disponible en línea https://www.birmingham.ac.uk/Documents/college-artslaw/corpus/Intro/Unit1.pdf Accedido 2018-06-09)

Goodfellow, I., Bengio, Y., y Courville, A. (2016). *Deep learning*. MIT Press. (http://www.deeplearningbook.org)

Hughes, M., Li, I., Kotoulas, S., y Suzumura, T. (2017). Medical text classification using convolutional neural networks. *Stud Health Technol Inform*, *235*, 246–50.

Kim, Y. (2014). Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*.

Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. En F. Pereira, C. J. C. Burges, L. Bottou, y K. Q. Weinberger (Eds.), *Advances in neural information processing systems 25* (pp. 1097–1105). Curran Associates, Inc.

Liu, B. (2012). *Sentiment analysis and opinion mining* (Vol. 5) (n.º 1). Morgan & Claypool

Publishers.

Manning, C. D., Raghavan, P., y Schütze, H. (2008). *Introduction to information retrieval*. Cambridge University Press.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., y Dean, J. (2013). Distributed representations of words and phrases and their compositionality. En C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, y K. Q. Weinberger (Eds.), *Advances in neural information processing systems 26* (pp. 3111–3119). Curran Associates, Inc. Descargado de http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

Mitchell, T. M. (1997). *Machine learning* (1.ª ed.). New York, NY, USA: McGraw-Hill, Inc.

Sinclair, J., y Wynne, M. (2005). Developing linguistic corpora: a guide to good practice. En (pp. 1–16). Oxford: Oxbow Books. (Disponible en línea http://ota.ox.ac.uk/documents/creating/dlc/ Accedido 2018-06-08)

Torrey, L., y Shavlik, J. (2009). Transfer learning. *Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques*, 242.

Vyrva, N. (2016). *Sentiment analysis in social media* (Tesis de Master no publicada). Ostfold University College.

Zhang, X., Zhao, J., y LeCun, Y. (2015). Character-level convolutional networks for text classification. En C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, y R. Garnett (Eds.), *Advances in neural information processing systems 28* (pp. 649–657). Curran Associates, Inc. Descargado de http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf