Aplicación para el análisis de sentimientos y tendencias en redes sociales

Universitat Politècnica de València - Campus d'Alcoi

Grado en Ingeniería Informática

Alumno:

Alejandro Pérez Sanjuán

Tutor:

Óscar Vega Gisbert

Junio de 2019

Resumen

Hoy en día las redes sociales son un pilar central de la comunicación social; aglutinan suficiente información como para entender los diferentes sesgos existentes respecto a temas de distinta índole. El objetivo de este trabajo es proveer de herramientas capaces de auxiliar en el análisis de los contextos sociales que surgen en dichas redes, utilizando para ello una combinación de diversas técnicas desarrolladas en el campo de las Ciencias de la Computación.

Palabras Clave: Sentimientos, redes sociales, aprendizaje máquina.

Abstract

Nowadays, social networks are a fundamental core of social communication; they bring together enough information to understand the different biases that exist on different topics. The objective of this work is to provide tools capable of assisting in the analysis of the social contexts that arise in these networks, using a combination of various techniques developed in the field of Computer Science.

Key Words: Sentiments, social networks, machine learning.

Dedicatorias

A mi familia.

Agradecimientos

Me gustaría expresar mi agradecimiento a mis profesores, tanto a los de la Escuela Politécnica Superior de Alcoy como a los de HU Hogeschool Utrecht; en especial a Aletta Smits, por compartir gustosa y amablemente sus muy extensos conocimientos sobre procesamiento del lenguaje natural, a Marieke Welle Donker-Kuijer, por sugerir el uso del coeficiente Kappa de Cohen y proveer material de ayuda, y a Óscar Vega Gisbert, por su paciencia y por aceptar ser mi tutor.

Índice

1. Introducción **9**
1.1. Objetivos 9
1.2. Secuenciación del proyecto 10
1.3. Motivación 12
1.4. Literatura relacionada 13
1.5. Metodología 14

2. Marco Teórico **15**
2.1. Fundamentos de Redes Neuronales 15
2.1.1. Definición 15
2.1.2. Neurona Artificial 15
2.1.3. Topologías en redes neuronales 17
2.2. Redes neuronales prealimentadas 17
2.2.1. Función de coste 19
2.2.2. Descenso del gradiente 20
2.2.3. Retropropagación 21
2.3. Redes neuronales recurrentes 22
2.3.1. Función de coste 24
2.3.2. Desvanecimiento del Gradiente 25
2.4. Redes LSTM 25

3. Desarrollo **27**
3.1. Los datos 27
3.1.1. Origen 27
3.1.2. Limpieza 28
3.1.3. Formato 28
3.2. Arquitectura de la red 29
3.2.1. Modelo 29
3.2.2. Función de coste 30
3.2.3. Funciones de activación 31
3.3. Características de la aplicación 31
3.3.1. Programación de la red 31
3.3.2. Interfaz Gráfica 31

4. Resultados **32**
4.1. Primeros Resultados 32
4.2. Interrupción temprana 34
4.3. Reducción de complejidad 34
4.4. Uso de vectores de palabras 35
4.5. Evaluación mediante el coeficiente Kappa 37

5. Conclusiones **40**
5.1. Conclusiones sobre los resultados 40
5.2. Conclusiones finales 40

Índice de figuras

1. Componentes básicos de una neurona artificial expresados en forma de grafo computacional [31]. 17
2. Arquitectura neuronal compuesta por tres capas. 18
3. Ejemplo de red neuronal con conexiones ponderadas entre las capas 0 y 1. 19
4. Ratio bajo (izquierda) frente a ratio alto (derecha). Fuente: [38]. 20
5. Red N neuronal recurrente. 24
6. Unidad LSTM. 26
7. Representación *one hot* para la palabra *ala* dado el diccionario D. 29
8. Distribución de las capas del modelo utilizado. 30
9. Vista parcial de la aplicación interactiva creada con *Dash Open Source*. El código se puede encontrar aquí. 32
10. Gráficas de los valores de coste para los conjuntos de datos de entrenamiento y validación. 33
11. Gráficas de los valores de precisión de la red ejecutando el conjunto de datos de entrenamiento y validación. 34
12. Gráficas de los valores de precisión de la red ejecutando el conjunto de datos de entrenamiento y validación, aplicando interrupción temprana. 35
13. Gráficas de los valores de precisión de la red ejecutando el conjunto de datos de entrenamiento y validación, aplicando reducción de complejidad 36
14. Gráficas de los valores de precisión de la red ejecutando el conjunto de datos de entrenamiento y validación, usando GloVe para mapear las palabras en vectores. 37
15. Gráficas de los valores de coste de la red ejecutando el conjunto de datos de entrenamiento y validación, usando GloVe para mapear las palabras en vectores. 38

Índice de cuadros

1. Plan para los meses octubre y noviembre. B se refiere a *búsqueda de la bibliografía* y L se refiere a *lectura de la bibliografía*. . 10
2. Plan para los meses diciembre y enero. PI se refiere a *primera implementación* y MI se refiere a *mejora de la implementación*. 10
3. Plan para el mes febrero. A se refiere a *ampliación de la bibliografía* y L se refiere a *lectura de la bibliografía*. 11
4. Plan para los meses marzo y abril. NI se refiere a *nueva implementación* y CI se refiere a *corrección de implementación*. . 11
5. Plan para los meses marzo y abril. DF se refiere a *borrador del documento*, R se refiere a *revisión* y E se refiere a *entrega*. . . 12
6. Tabla con los resultados obtenidos en 5 ejecuciones del modelo para el conjunto de datos *test*. 39

1. Introducción

1.1. Objetivos

Los objetivos de este trabajo se exponen a continuación:

- Presentar un trabajo que cumpla los estándares académicos de calidad.
- Introducirse en un campo no estudiado anteriormente, utilizando los conocimientos aprendidos a lo largo del grado.
- Entender los aspectos de implementación de redes neuronales.
- Entender el marco teórico subyacente tras las redes neuronales.
- Proporcionar una herramienta de análisis de datos fiable basada en inteligencia artificial.
- Adquirir competencia en los campos *aprendizaje máquina*, *aprendizaje profundo*, *aprendizaje supervisado* y *ciencia de datos*.
- Identificar los problemas y soluciones asociados a los distintos tipos de redes neuronales.
- Conocer, y entender, al menos una de las librerías de inteligencia artificial que se usan en el mundo laboral y académico.
- Entender y aplicar conceptos sobre procesamiento del lenguaje natural.
- Entender y aplicar los modelos *one hot*, *GloVe* y *word2vec* para representar palabras en forma de vectores.
- Realizar un bibliografía formal y adecuada que contenga recursos tradicionales (publicaciones científicas y libros) y recursos nuevos (blogs, vídeos, etcétera).
- Diseñar e implementar una interfaz de usuario para mostrar los resultados del trabajo, usando para ello una librería *open source*.

1.2. Secuenciación del proyecto

La secuenciación seguida para realizar este trabajo, junto con el plan semanal de cada mes, se describe a continuación:

1. **Investigación:** (cuadro 1) consiste en la búsqueda y lectura de la bibliografía necesaria para realizar el proyecto.

	Octubre	**Noviembre**
Semana 1	B	B
Semana 2	B	B y L
Semana 3	B	B y L
Semana 4	B	B y L

Cuadro 1: Plan para los meses octubre y noviembre. B se refiere a *búsqueda de la bibliografía* y L se refiere a *lectura de la bibliografía*.

   Se puede observar que, en algunas semanas, la búsqueda y la lectura coinciden; esto se debe a que la lectura de la bibliografía conduce a nuevos materiales y recursos.

2. **Primera implementación:** (cuadro 2) se programa una primera versión de la red neuronal, usando para ello *Python* y la librería *numpy*.

	Diciembre	**Enero**
Semana 1	PI	-
Semana 2	PI	-
Semana 3	PI	MI
Semana 4	-	MI

Cuadro 2: Plan para los meses diciembre y enero. PI se refiere a *primera implementación* y MI se refiere a *mejora de la implementación*.

   La primera versión de la red no funcionaba acorde al modelo teórico, de forma que se tuvo que corregir y mejorar.

3. **Ampliación de bibliografía:** (cuadro 3) tras implementar la primera versión y observar los problemas, se considera apropiado investigar más.

	Febrero
Semana 1	A
Semana 2	A
Semana 3	A y L
Semana 4	A y L

Cuadro 3: Plan para el mes febrero. A se refiere a *ampliación de la bibliografía* y L se refiere a *lectura de la bibliografía*.

4. **Nueva implementación:** (cuadro 4) se implementa la red utilizando la librería *TensorFlow*.

	Marzo	**Abril**
Semana 1	NI	NI
Semana 2	NI	CI
Semana 3	NI	CI
Semana 4	NI	CI

Cuadro 4: Plan para los meses marzo y abril. NI se refiere a *nueva implementación* y CI se refiere a *corrección de implementación*.

   Tras implementar la red, se observaron varios errores de funcionamiento que se subsanaron las semanas siguientes.

5. **Interfaz de usuario, documentación y revisión:** (cuadro 5) se elabora la documentación final utilizando para ello todos las notas y apuntes creados hasta la fecha. También se crea una interfaz de usuario simple que permita interactuar con la red de forma intuitiva. La fase acaba con la entrega del trabajo.

	Mayo	**Junio**
Semana 1	BD	E
Semana 2	BD	-
Semana 3	BD	-
Semana 4	R	-

Cuadro 5: Plan para los meses marzo y abril. DF se refiere a *borrador del documento*, R se refiere a *revisión* y E se refiere a *entrega*.

1.3. Motivación

El *análisis de sentimientos*, también conocido como *minería de opinión*, es el proceso de extraer información subjetiva utilizando diferentes herramientas y técnicas.

Aunque en los últimos años esta técnica se ha hecho muy popular, sus orígenes se remontan a finales del siglo XX, con el análisis subjetivo que la comunidad de lingüística computacional llevaba a cabo en los años 90 [23].

Anteriormente a la irrupción de internet, no habían muchos estudios acerca de este tema debido a la limitada disponibilidad de datos para realizar análisis [34]. Sin embargo, es posible argumentar de forma lógica que la opinión de otros siempre ha sido un tema relevante, especialmente para aquellas personas que ostenten una posición de poder. El libro *1984*, de George Orwell, plantea esta posibilidad y su utilización para dañar a la sociedad [33]

Hoy, con el crecimiento explosivo de las redes sociales, el análisis de sentimientos es más relevante que nunca. Sus muchas aplicaciones crean un debate permanente sobre las diferentes aproximaciones teóricas; además, la minería de opinión es una característica cada vez más demandada en compañías de todo el mundo que quieren analizar los detalles de sus distintos procesos de negocio. Algunas aplicaciones podrían ser [6]:

1. **Opinión de los empleados:** Puede ayudar a una compañía a realizar un seguimiento de los sentimientos, opiniones y pensamientos de los empleados para con sus trabajos.

2. **Experiencia de cliente:** Una empresa puede utilizar las opiniones de sus clientes para mejorar la experiencia de compra, crear nuevos productos o mejorar algunas ya existentes.

3. **Inteligencia de negocios:** El *business intelligence* se utiliza para estimar el impacto que las decisiones y/o acciones de una compañía tendrán en su futuro. El análisis de sentimientos permite utilizar datos del pasado para predecir el futuro.

Estas son sólo algunas aplicaciones del análisis de sentimientos, pero sirven bien para ilustrar las muchas maneras en las que uno puede utilizar esta técnica.

Este trabajo se centrará en la aproximación con la que el *aprendizaje máquina*, o *machine learning*, trata la minería de opinión, usando para ello una de las técnica más potentes a día de hoy para ello: las *redes neuronales LSTM*.

1.4. Literatura relacionada

Internet conlleva un crecimiento exponencial de la información contenida en él. Herramientas como *Google* permiten encontrar una cantidad de información enorme, pero esta debe ser filtrada. A continuación se exponen una serie de recursos que, tras una exhaustiva revisión, se consideran fiables y han sido utlizados para realizar este trabajo. Los recursos no sólo contienen los tradicionales *papers* y libros de alto contenido académico sino que, además, incluyen cursos, blogs y vídeos de alto valor y con un nivel de formalidad más que aceptable.

Creado y gestionado por Christopher Olah, *colah's blog* [32] contiene explicaciones accesibles para temas muy complejos, como LSTM o *grafos computacionales*. Olah es un investigador que trabaja en *Google Brain*; en la elaboración de sus post utiliza una revisión por pares del contenido que publica.

Otro recurso online de gran utilidad es *YouTube*, donde uno puede encontrar cursos de procesamiento del lenguaje natural [2] impartidos por la universidad de Stanford. Junto con la plataforma de aprendizaje online *Cursera* y su curso en inteligencia artificial y aprendizaje máquina impartido por Andrew NG [29], forman una poderosa herramienta que facilita el aprendizaje.

En cuanto a publicaciones científicas, hay miles de las que poder aprender algo. Aquí se exponen las que han resultado ser más útiles para la elaboración de este trabajo. Empezando por la publicación original en la que Hochreiter

y Schmidhuber [18] presentaron por primera vez qué es una red neuronal recurrente LSTM.

En [23] se expone la correlación entre el crecimiento de internet y el crecimiento del análisis de sentimientos. Los autores explican y analizan esta correlación presentando una taxonomía de los temas de investigación en distintos campos. La conclusión que parecen extraer es la de que hay una correlación fuerte entre el número de publicaciones científicas en el ámbito de la minería de opinión y el crecimiento de internet.

Una interesante perspectiva del proceso de entrenamiento y por qué es complejo puede ser encontrado en [13], donde los autores estudian por qué el *descenso del gradiente* con una inicialización de pesos y umbrales aleatoria da resultados pobres. Una inicialización aleatoria de estos parámetros puede conducir a problemas para encontrar un mínimo adecuado para la función de coste, lo que provoca a su vez que los valores de dichos parámetros no puedan ser actualizados de forma óptima. Este artículo resulta un interesante complemento para [7], donde se expone la complejidad del aprendizaje de dependencias de datos separadas en el tiempo por una distancia suficientemente grande.

Para publicaciones científicas más específicas acerca del tema, los recursos [27], [5] y [20] han sido de gran utilidad en la elaboración de este trabajo. Los tres aplican técnicas de aprendizaje máquina para realizar análisis de sentimientos. Específicamente, utilizan redes neuronales recurrentes LSTM para mejorar la conexión entre las dependencias de datos.

En cuanto a libros, [16], [36] y [37] se han utilizado en la elaboración de este texto. Mención especial para el libro de Raúl Rojas [37], que provee de un enfoque elegante y de alto nivel sobre la materia.

Los recursos anteriormente citados no son los únicos que se han usado, pero sí los más relevantes. Durante el resto del trabajo se citarán nuevos, variados y relevantes recursos.

1.5. Metodología

Sin tener en cuenta la introducción, el trabajo se estructura en tres partes. En la primera parte se establecerán las bases teóricas necesarias sobre las que se sustenta la práctica. Una vez expuesto el contexto teórico, se procederá a desarrollar la arquitectura de la aplicación, explicando sus distintos niveles y la relación entre ellos. Por último, la tercera parte constará de la exposición de resultados.

Cabe señalar que este trabajo no conforma un texto del estado del arte. Algunas definiciones y requerimientos matemáticos se han omitido en pos de la operatividad.

2. Marco Teórico

2.1. Fundamentos de Redes Neuronales

Se puede decir que las redes neuronales nacieron bajo la tutela de la *biomímesis*, la ciencia que estudia y mimetiza la naturaleza para resolver problemas [10].

El inicio de este campo de estudio se produjo en 1943. *Warren McCulloch* y *Walter Pitts* [26] escribieron un artículo sobre el posible funcionamiento de las neuronas biológicas.

Con el desarrollo de la arquitectura *von Neumann*, el interés de los estudiosos se centró en la computación convencional, dejando de lado las redes neuronales. No fue hasta 1986 [25] que se progresó en el campo hasta alcanzar un modelo de neurona multicapa.

Hoy en día, las redes neuronales se usan en todo tipo de aplicaciones y dispositivos.

2.1.1. Definición

Esencialmente, una red neuronal artificial es un modelo matemático que imita algunas características del funcionamiento de las redes neuronales biológicas. Formalmente, se puede describir una red neuronal artificial como una función f que mapea una determinada entrada x en una determinada salida y.

$$f : x \longrightarrow y \tag{1}$$

Las redes neuronales están formadas por un conjunto de unidades simples de procesamiento llamadas *neuronas*. Las neuronas se comunican entre ellas mediante el envío de señales a través de un número determinado de conexiones ponderadas.

2.1.2. Neurona Artificial

Como se ha comentado, las *neuronas* son unidades simples de procesamiento. En este trabajo se presentan las neuronas artificiales según el modelo

estándar descrito por *Rumelhart* y *McClelland* [25]. Una neurona artificial consiste en:

1. Un conjunto de entradas x_j con $j = 1, \ldots, n$ y una salida y_j.

2. Un conjunto de pesos o ponderaciones w_{jk} con $k = 1, \ldots, n$ llamados *pesos sinápticos*.

3. Una regla de propagación s_j definida a partir del conjunto de entradas y de los pesos sinápticos:

$$s_j = \sum_{j=1}^{n} w_{jk} x_j \tag{2}$$

4. Un sesgo o umbral θ_j, a partir del cuál la neurona queda activa. Añadiendo el sesgo, la regla de propagación queda de la forma:

$$s_j = \sum_{j=1}^{n} w_{jk} x_j + \theta_j \tag{3}$$

5. Una función de activación que sirve para modificar la s_j de forma no lineal. Esto es necesario porque cada neurona representa un modelo de regresión lineal, y la suma de n regresiones lineales es equivalente a otra regresión lineal, reduciendo drásticamente el espacio de problemas que se pueden modelizar con la red.

   La *función sigmoide* es la más común en los textos académicos que explican los fundamentes de redes neuronales.

$$\sigma(s) = \frac{1}{1 + e^{-s}} \tag{4}$$

   Aunque existen otras [30] [1], las bases teóricas aquí expuestas harán uso de dicha función. En la parte de desarrollo, se especifican las funciones utilizadas para el modelo real y el porqué de su elección.

Con las propiedades enumeradas anteriormente, el modelo de neurona estándar se puede presentar de la siguiente forma (figura 1):

$$y_j = \sigma\left(\sum_{j=1}^{n} w_{jk} x_j + \theta_j\right) \tag{5}$$

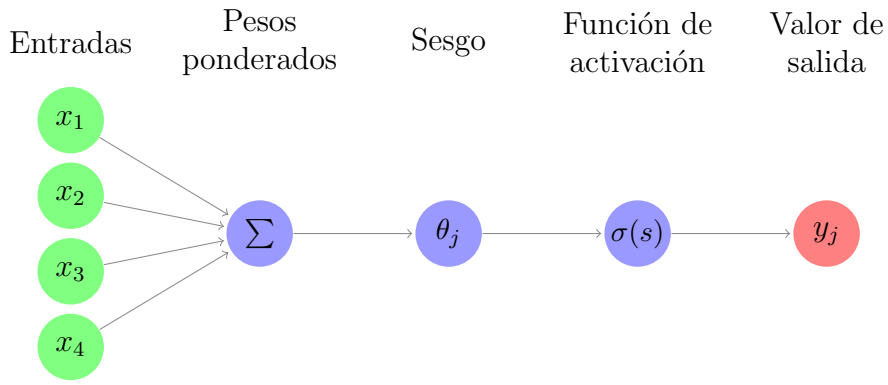

Figura 1: Componentes básicos de una neurona artificial expresados en forma de grafo computacional [31].

2.1.3. Topologías en redes neuronales

En una red neuronal los nodos se conectan mediante sinapsis. El patrón de conexión determina el comportamiento de la red. Típicamente, las neuronas se agrupan en un nivel topológico superior denominado *capa*. Se considera que una red neuronal está formada por al menos una capa. En este trabajo sólo se considerarán 2 tipos de redes neuronales:

1. *Prealimentadas*: son aquellas en las que las neuronas de una determinada capa i sólo pueden transferir información a la capa $i + 1$.

2. *Recurrentes*: son aquellas que pueden contener conexiones de retroalimentación entre las neuronas.

2.2. Redes neuronales prealimentadas

Conocidas como redes *feedforward*. En ellas, la información se propaga desde las capas previas hasta las capas posteriores, sin conexiones de retroalimentación.

En las redes prealimentadas, se distinguen 3 tipos de capas (figura 2):

1. *Capa de entrada*: provee información del exterior a la red neuronal.

2. *Capa oculta*: se llaman de esa forma debido a que no interactúan de forma directa con el entorno exterior.

3. *Capa de salida*: se utiliza para transferir la información computada al exterior.

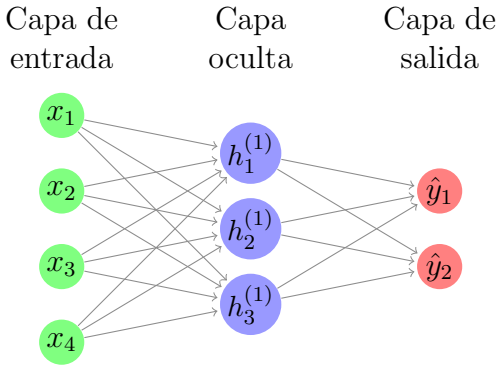

Figura 2: Arquitectura neuronal compuesta por tres capas.

Representar la red neuronal usando la ecuación (5) para cada neurona puede resultar engorroso, así que se suele usar la forma matricial para representar la red por capas.

Sea l la capa que se quiere representar, la forma matricial para l se puede describir como:

$$y^{[l]} = w^{[l]} x^{[l-1]} + \theta^{[l]} \tag{6}$$

Donde las dimensiones de cada parámetro se definen en función del número de neuronas n de la capa correspondiente:

1. $y^{[l]} \rightarrow [n^{[l]}, 1]$
2. $w^{[l]} \rightarrow [n^{[l]}, n^{[l-1]}]$
3. $x^{[l-1]} \rightarrow [n^{[l-1]}, 1]$
4. $\theta^{[l]} \rightarrow [n^{[l]}, 1]$

De los parámetros anteriores, la matriz de pesos tiene especial relevancia por la notación de las ponderaciones individuales que conectan las capas. Así, una conexión se define por $w_{jk}^{[l]}$, donde j es la neurona destino de la capa l y k es la neurona origen de la capa $l-1$.

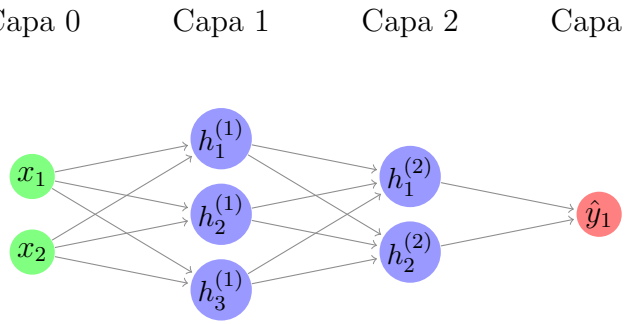

Figura 3: Ejemplo de red neuronal con conexiones ponderadas entre las capas 0 y 1.

Para la figura 3, la matriz de pesos de la capa 1 se definiría como:

$$w^{[1]} = \begin{bmatrix} w_{11}^{[1]} & w_{12}^{[1]} \\ w_{21}^{[1]} & w_{22}^{[1]} \\ w_{31}^{[1]} & w_{32}^{[1]} \end{bmatrix} \tag{7}$$

2.2.1. Función de coste

El funcionamiento de una red neuronal artificial depende de forma directa de los valores de las ponderaciones y de los sesgos. Cambiando estos valores se modula el comportamiento de la red. Es necesario pues, evaluar la idoneidad de dichos valores mediante una función de coste.

Al igual que con la función sigmoide, el *error cuadrático medio* es la función típicamente elegida en los textos académicos que explican redes neuronales; por tanto, de forma análoga, se escogerá esta función para exponer la teoría y más adelante se especificará la función escogida para el modelo real.

$$J(w_{jk}, \theta_j) = \frac{1}{2} \sum_{i=1}^{n} (\bar{y}_i - y_i)^2 \tag{8}$$

Donde $\bar{y}$ es el resultado de la red neuronal artificial e y_i es el resultado real. Cuanto más próximo sea el valor de J a cero, mejores serán las predicciones de la red neuronal.

2.2.2. Descenso del gradiente

Descenso del Gradiente es un algoritmo iterativo que se usa para obtener un mínimo local, bajo ciertas condiciones, global, de una función.

Por lo general, minimizar una función $f(x_1, x_2, \ldots, x_i)$ consiste en alterar los valores x_i siguiendo un criterio de reducción para el valor de f. Esto es, se empieza con una asunción inicial para x_i, minimizando los valores de x_i de forma progresiva para reducir $f(x_1, x_2, \ldots, x_i)$. El núcleo del algoritmo radica en la siguiente expresión:

$$x_i = x_i - \eta \cdot \nabla f(x_1, x_2, \ldots, x_i) \tag{9}$$

Donde η es el ratio de aprendizaje. Un ratio de aprendizaje demasiado bajo dará como resultado un número excesivo de pasos para alcanzar el mínimo. Por el contrario, un ratio demasiado alto hará que los pasos sean muy grandes, dificultando la búsqueda del mínimo local (figura 4).

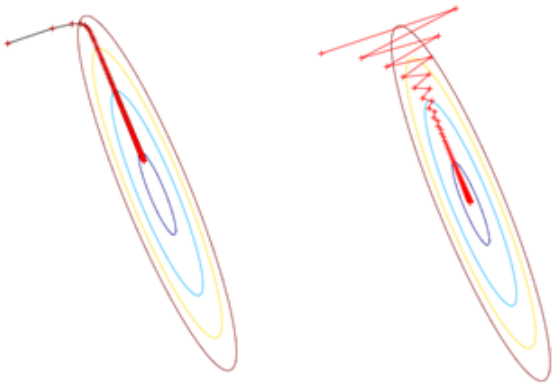

Figura 4: Ratio bajo (izquierda) frente a ratio alto (derecha). Fuente: [38].

La operación $\nabla f(x_1, x_2, \ldots, x_i)$ corresponde al gradiente, que se expresa de la siguiente forma:

$$\nabla f(x_1, x_2, \ldots, x_i) = \frac{\partial}{\partial x_i} f(x_1, x_2, \ldots, x_i) = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_i} \end{bmatrix} \tag{10}$$

Donde $f(x_1, x_2, \ldots, x_i)$ es una función derivable.

Con todo, el algoritmo descenso del gradiente se puede describir como se presenta en el algoritmo 1.

Algoritmo 1 - Descenso del gradiente

1: **for** $i = 1$ *to* I **do**
2: $\nabla f(x_1, x_2, \ldots, x_i)$
3: $x_i \leftarrow x_i - \eta \cdot \nabla f(x_1, x_2, \ldots, x_i)$
4: **end for**
5: *return* x_i

2.2.3. Retropropagación

El algoritmo de retropropagación es uno de los métodos de entrenamiento más comunes en redes neuronales. En esencia, el algoritmo de retropropagación permite calcular el error de la red, y corregir sus parámetros en función de dicho error.

Con todas las ecuaciones presentadas, la composición de funciones para la capa L se describe como:

$$J\left(y_j^{[L]}\left(s_j^{[L]}\left(w, \theta\right)\right)\right) \tag{11}$$

Para obtener la variación que se produce en J si variamos los pesos, se usa la regla de la cadena:

$$\frac{\partial J}{\partial w_{jk}^{[L]}} = \frac{\partial J}{\partial y_j^{[L]}} \cdot \frac{\partial y_j^{[L]}}{\partial s_j^{[L]}} \cdot \frac{\partial s_j^{[L]}}{\partial w_{jk}^{[L]}} \tag{12}$$

El mismo proceso se sigue si se quiere obtener la variación de J respecto de los sesgos:

$$\frac{\partial J}{\partial \theta_{jk}^{[L]}} = \frac{\partial J}{\partial y_j^{[L]}} \cdot \frac{\partial y_j^{[L]}}{\partial s_j^{[L]}} \cdot \frac{\partial s_j^{[L]}}{\partial \theta_{jk}^{[L]}} \tag{13}$$

Nótese que hay términos comunes en las ecuaciones (12) y (13). Dichos términos representan el error imputado a una neurona de la capa L.

$$\delta^{[L]} = \frac{\partial J}{\partial s_j^{[L]}} = \frac{\partial J}{\partial y_j^{[L]}} \cdot \frac{\partial y_j^{[L]}}{\partial s_j^{[L]}} \tag{14}$$

Ahora se procederá a desglosar los términos que aparecen en las expresiones anteriores, con el fin de obtener el gradiente de la función en la última capa. Empezando por la derivada del error cuadrático medio.

$$\frac{\partial J}{\partial y_j^{[L]}} = \frac{2}{n}\sum_{i=1}^{n}(\bar{y}_i - y_i) \tag{15}$$

La variación de la activación con respecto de la suma ponderada:

$$\frac{\partial y_j^{[L]}}{\partial s_j^{[L]}} = \sigma'\left(s_j^{[L]}\right) = \sigma\left(s_j^{[L]}\right) \cdot \left(1 - \sigma\left(s_j^{[L]}\right)\right) \tag{16}$$

La alteración de la suma ponderada con respecto a los pesos:

$$\frac{\partial s_j^{[L]}}{\partial w_{jk}^{[L]}} = y_j^{[L-1]} \tag{17}$$

Y con respecto a los sesgos:

$$\frac{\partial s_j^{[L]}}{\partial \theta_j^{[L]}} = 1 \tag{18}$$

Finalmente, las expresiones que permiten conocer el gradiente quedan de la forma:

$$\frac{\partial J}{\partial w_{jk}^{[L]}} = \delta^{[L]} \cdot y_j^{[L-1]} \tag{19}$$

$$\frac{\partial J}{\partial \theta_j^{[L]}} = \delta^{[L]} \tag{20}$$

En base a las ecuaciones descritas, se presenta en el algoritmo 2 el proceso de retropropagación, en notación matricial, junto con el descenso del gradiente que corrige los parámetros internos de la red.

2.3. Redes neuronales recurrentes

Cuando se trata de procesar lenguaje natural, las redes neuronales convencionales (*feedforward*) no son apropiadas. Esto se debe a la necesidad

Algoritmo 2 - Retropropagación

for 1 *to* L **do**
    $y^{[l]} = w^{[l]}x^{[l-1]} + \theta^{[l]}$
end for
$\delta^{[L]} = \dfrac{\partial J}{\partial y^{[L]}} \cdot \dfrac{\partial y^{[L]}}{\partial s^{[L]}}$
for L *to* 1 **do**
    $\dfrac{\partial J}{\partial w^{[l]}} = y^{[l+1]} \cdot \delta^{[l]}$
    $\dfrac{\partial J}{\partial \theta^{[l]}} = \delta^{[l]}$
    $\delta^{[l]} = \left(\left(w^{l+1}\right)^T \cdot \delta^{[l+1]}\right) \odot \sigma'\left(s^l\right)$
    $w^{[l]} = w^{[l]} - \eta \cdot \delta^{[l]} \cdot \left(y^{[l-1]}\right)^T$
    $\theta^{[l]} = \theta^{[l]} - \eta \cdot \delta^{[l]}$
end for

de tener una entrada variable para la red. Las redes neuronales recurrentes solucionan este problema usando bucles de retroalimentación.

Una red neuronal recurrente, también llamada *feedback*, es aquella que contiene bucles de retroalimentación, permitiendo que la información adquiera carácter persistente. Estas redes son adecuadas para procesar *secuencias*. Una *secuencia* es un conjunto de datos ordenados. Las redes recurrentes constituyen una función capaz de mapear los siguientes casos [2]:

1. $f : secuencia \longrightarrow \mathbb{R}^L$
2. $f : \mathbb{R}^L \longrightarrow secuencia$
3. $f : secuencia \longrightarrow secuencia$

Donde $\mathbb{R}^L$ representa un vector de longitud L.

El funcionamiento de una red recurrente añade una variable t nueva, que por convenio recibe el nombre de *tiempo* a pesar de que no tiene que referenciar a la magnitud física que recibe el mismo nombre. La red recurrente se "desdobla" en función de los instantes de tiempo (figura 5).

Las neuronas de una red recurrente cambian sus propiedades respecto a una red *feedforward*.

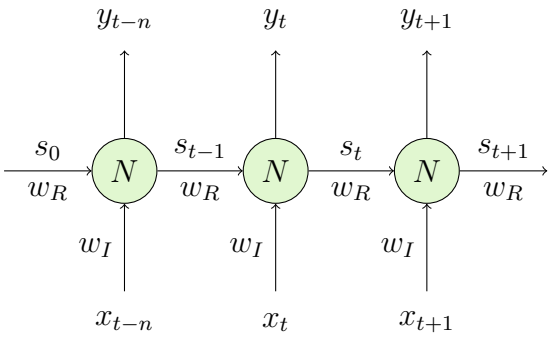

Figura 5: Red N neuronal recurrente.

1. En lugar de una matriz de pesos, hay dos: w_I y w_R. Tanto w_I como w_R se utilizan en todos los instantes t. Esto es, por cada instante t no se genera una nueva matriz de pesos.

2. La suma ponderada cambia para tener en cuenta dichas matrices de pesos.
$$s_t = w_I \cdot x_t + w_R \cdot s_{t-1} + \theta_t \tag{21}$$

3. La salida final de la red corresponde a la siguiente expresión:
$$y_t = \sigma\left(w_I \cdot x_t + w_R \cdot s_{t-1} + \theta_t\right) \tag{22}$$

2.3.1. Función de coste

Con la red "desdoblada", se procede a obtener el gradiente para realizar *retropropagación* de forma adecuada. Se puede observar que hay múltiples costes; en concreto, hay un coste por cada instante t. Los gradientes deben combinarse para obtener los parámetros de forma adecuada. Por ejemplo, el coste respecto de w_I se expresa como:

$$\frac{\partial J}{\partial w_I} = \sum_{t=1}^{n} \frac{\partial J_t}{\partial w_I} \tag{23}$$

El resto de ecuaciones, modificadas por esta nueva condición, quedan como:

$$\frac{\partial J}{\partial w_R} = \sum_{t=1}^{n} \frac{\partial J_t}{\partial w_R} \tag{24}$$

$$\frac{\partial J}{\partial \theta_t} = \sum_{t=1}^{n} \frac{\partial J_t}{\partial \theta_t} \tag{25}$$

2.3.2. Desvanecimiento del Gradiente

Supóngase una red recurrente tal que $t = 3$. La salida de dicha red se expresaría como:

$$y_3 = \sigma(w_I \cdot x_3 + w_R \cdot (w_I \cdot x_2 + w_R \cdot (w_I \cdot x_1 + w_R \cdot x_0)) \tag{26}$$

Si se evalúa el coste de dicha red para obtener el gradiente, la fórmula obtenida queda como:

$$\frac{\partial J_3}{\partial w_I} = \frac{\partial J_3}{\partial y_3} \cdot \frac{\partial y_3}{\partial s_3} \cdot \frac{\partial s_3}{\partial s_2} \cdot \frac{\partial s_2}{\partial s_1} \cdot \frac{\partial s_1}{\partial w_I} \tag{27}$$

De las ecuaciones anteriores se deduce fácilmente que las fórmulas aumentan proporcionalmente al tamaño de la red. Los valores que forman los distintos términos de la red son pequeños debido a la función de activación y, al multiplicarse, tienden a 0 conforme la red aumenta de tamaño. Esto significa que los pesos y los sesgos no pueden actualizarse correctamente porque el gradiente que actúa de corrector se desvanece.

Este problema se conoce como *desvanecimiento del gradiente* [17], y provoca que la red sea incapaz de conectar dependencias de información en entradas de datos suficientemente grandes. Con el *desvanecimiento del gradiente* la información procesada por la red pierde cohesión, y los resultados tienden a ser incorrectos.

Aunque existen varias soluciones a este problema, este trabajo se enfocará en una propuesta por Sepp Hochreiter y Jürgen Schmidhuber en 1997 [18]: las redes *LSTM*.

2.4. Redes LSTM

Las redes *LSTM* (*Long short-term memory*) son un tipo especial de red recurrente diseñadas específicamente para evitar el problema del *desvanecimiento del gradiente*.

Las redes *LSTM* se componen de una secuencia de unidades, o celdas, encadenadas. La estructura de las unidades es idéntica (figura 6), pero no los

valores que almacenan en forma de vector. Por cada instante de tiempo t, un conjunto de vectores es procesado:

1. Una puerta para descartar: $f_t = \sigma\left(W_f \times x_t + U_f \times h_{t-1} + b_f\right)$
2. Una puerta de entrada: $i_t = \sigma\left(W_i \times x_t + U_i \times h_{t-1} + b_i\right)$
3. Un vector de nuevos candidatos para el estado de la unidad: $\tilde{C}_t = tanh\left(W_C \times x_t + U_C \times h_{t-1} + b_C\right)$
4. Una puerta de salida: $o_t = \sigma\left(W_o \times x_t + U_o \times h_{t-1} + b_o\right)$
5. La memoria de la unidad: $C_t = i_t \times \tilde{C}_t + f_t \times C_{t-1}$
6. Una capa de salida oculta: $h_t = o_t \times tanh(C_t)$

Donde $W_f, U_f, W_i, U_i,$ son matrices de pesos y b_f, b_i, b_C, b_0 vectores de sesgos.

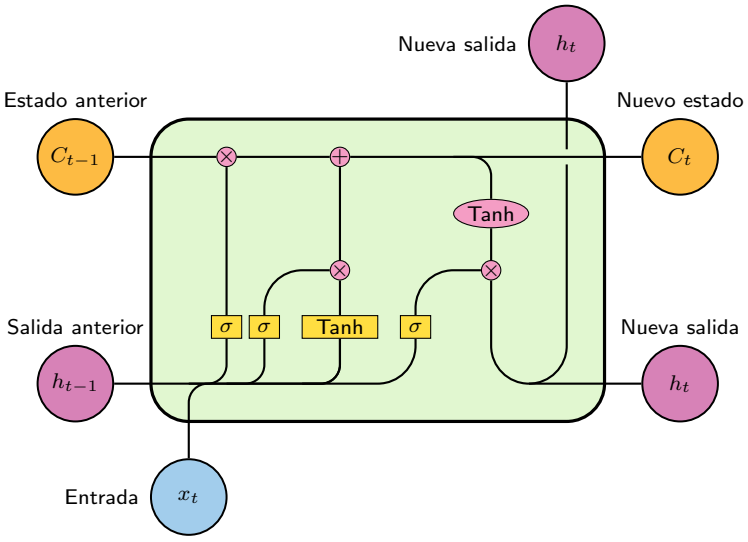

Figura 6: Unidad LSTM.

El proceso interno se describe como sigue [32]:

1. *Paso 1:* La unidad decide qué información va a ser descartada. Esto se hace computando f_t. Una vez ocurre esto, un número entre 0 y 1 se crea para cada número en el estado de la celda C_{t-1}, donde un 1 representa 'mantener completamente' y un 0 representa 'olvidar completamente'.

2. *Paso 2:* La celda decide qué información se va a almacenar, usando $\tilde{C}_t$ y i_t.

3. *Paso 3:* La unidad actualiza el antiguo estado C_{t-1} con el nuevo C_t. Esto se hace computando la fórmula C_t

4. *Paso 4:* La salida h_t es determinada en función de una versión filtrada del estado de la celda.

3. Desarrollo

En esta sección se expone el proceso seguido para transformar los datos a un formato entendible por un modelo de aprendizaje máquina y se explican las distintas iteraciones seguidas en la creación de dicho modelo, detallando los problemas asociados y las soluciones encontradas para éstos.

3.1. Los datos

3.1.1. Origen

Para entrenar a la red neuronal hace falta una cantidad considerable de datos. En este caso, los datos están compuestos por 1,6 millones de tweets, y son proporcionados por la Universidad de Stanford [40]; en concreto, los datos tienen su origen en el proyecto *open source Sentiment140*.

El conjunto de datos contiene, además de los tweets, la polaridad asociada a estos; siendo 0 ó 4 las posibles opciones para negativo y positivo respectivamente. Esta polaridad se ha computado usando algoritmos de aprendizaje no supervisado, como se explica en [15].

Este conjunto de datos se ha elegido por tres motivos:

1. La cantidad de datos es suficientemente grande como para permitir extraer conclusiones relevantes.

2. La fuente que proporciona los datos es fiable.

3. Los datos están en inglés, que es un idioma gramaticalmente menos complejo que el castellano.

3.1.2. Limpieza

La limpieza de los datos es uno de los pasos más importantes a tener en cuenta en este tipo de aplicaciones. En este caso particular, los datos contienen mucho ruido y se necesita aplicar una serie de filtros y técnicas para eliminar la información que no resulta útil.

Los pasos que se han seguido para limpiar los datos se detallan a continuación:

1. Convertir el texto a minúscula.
2. Eliminar las menciones a usuarios, los *hashtags* y los enlaces a páginas web.
3. Eliminar todos los caracteres alfanuméricos, así como todos los números y otros caracteres especiales.
4. Eliminar palabras vacías, como artículos, con la ayuda de un diccionario inglés.
5. Corregir errores gramaticales en el texto restante.
6. Los tweets cuya longitud sea 0 como resultado de la limpieza serán descartados.

3.1.3. Formato

Existen varios modelos [28] [35] para representar palabras en el campo del procesamiento del lenguaje natural; en las primeras versiones del modelo, se ha utilizado la técnica conocida como codificación *one hot*. Más adelante se exploran otras opciones

Esta técnica consiste en crear, o cargar, un diccionario D de palabras y representar cada palabra como un vector de ceros y un uno en el índice correspondiente a dicha palabra en D (figura 7).

La representación puede contraerse aún más. Cada palabra puede representarse simplemente por un número entero, que indica el índice de la posición en la que se encuentra la palabra en el diccionario D. Así pues, suponiendo que D tiene una longitud de 3000, el número 2999 equivale a la palabra *zuzón* si el indexado empieza en 0.

Para modelizar un tweet completo, se concatenan los números que representan las palabras que lo forman en un sólo vector. Si el vector no tiene la

$$D = \begin{bmatrix} Ala \\ \vdots \\ Boli \\ \vdots \\ Zuzón \end{bmatrix} \; ; \quad Ala = \begin{bmatrix} 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{28}$$

Figura 7: Representación *one hot* para la palabra *ala* dado el diccionario D.

longitud necesaria, se completará el vector añadiendo elementos con valor 0 al mismo.

La longitud del diccionario se obtuvo tras recorrer todo el conjunto de datos, con un doble bucle, y anotar las palabras y su frecuencia. La longitud equivale al número de palabras con una frecuencia superior a 4. Esto se hizo así para reducir el coste computacional de la red, tanto en memoria como en tiempo de procesamiento. Tras la limpieza de datos, para este conjunto, la longitud del diccionario real es de 429.547 palabras, mientras que si se aplica la condición de la frecuencia mayor a 4, es de 67.251.

Finalmente, el conjunto de datos se dividió, con un muestreo aleatorio, en dos subconjuntos, uno de entrenamiento y otro de validación, con una proporción de 0.8 y 0.2 respectivamente. Esta técnica se conoce como *validación cruzada* [19] [4], y se utiliza para estudiar la posible aparición de problemas en el entrenamiento, como *sobreajuste* o *subajuste*. Más adelante se ampliará este tema.

3.2. Arquitectura de la red

3.2.1. Modelo

Para este modelo se ha optado por una arquitectura de 4 capas (figura 8). La primera capa es una capa de *word embeddings*, o incrustaciones de palabras, que sirve para transformar los tweets a vectores de números reales. La segunda capa consiste en 32 unidades LSTM que se encargarán de procesar dependencias de datos de forma eficiente. La tercera capa estará formada por 16 neuronas estándar, y servirá para añadir un nivel de complejidad al aprendizaje de la red. Por último, la capa número 4 será la encargada de

transmitir los resultados de la red; consiste en 2 neuronas que representan las dos posibles polaridades asociadas a los tweets.

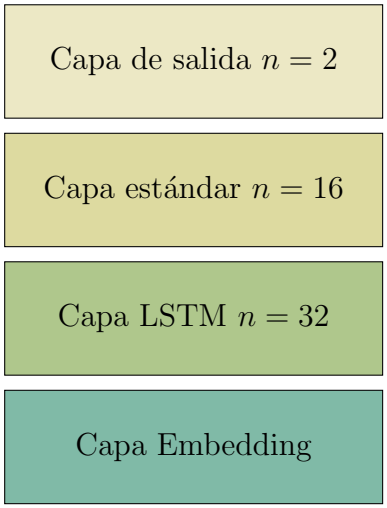

Figura 8: Distribución de las capas del modelo utilizado.

3.2.2. Función de coste

Como se ha clarificado en los fundamentos teóricos, el error cuadrático medio se ha utilizado para mostrar el funcionamiento de una red neuronal. En el modelo real, sin embargo, se ha usado *entropía cruzada binaria*:

$$H_q(q) = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i)) \tag{29}$$

donde y es la clase (0 y 4 en este caso) y $p(y)$ es la probabilidad de que la predicción sea 4.

La función *entropía cruzada* es la que se escoge típicamente en tareas de clasificación. En este caso, la clasificación es binaria y, por tanto, se aplica *entropía cruzada binaria*.

3.2.3. Funciones de activación

Además de la mencionada función sigmoide, este modelo hace uso de dos funciones más. La función $tanh(x)$ se aplica en las unidades LSTM:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{30}$$

La función *ReLU*, *rectificador*, se ha utilizado en la capa estánda de 16 neuronas:

$$ReLU(x) = max(0, x) \tag{31}$$

La función $tanh(x)$ es una de las usadas en el modelo LSTM para definir su comportamiento [18]. En cuanto a la función *ReLU* (rectificador lineal), en [14] se prueba que el uso de esta función mejora, en el campo del análisis de sentimientos, los resultados respecto a un modelo que use otras funciones para las capas de la red.

3.3. Características de la aplicación

3.3.1. Programación de la red

La instancia más primitiva de la programación de la red se creó utilizando PYTHON junto con la librería *numpy*. El código se puede encontrar aquí. Funcionaba de forma correcta, pero su rendimiento no era suficiente, puesto que 500 iteraciones sobre un conjunto de datos de longitud 50 tomaba alrededor de 5 minutos. Dado que el conjunto usado es de 1.6 millones entradas, el programa no era apropiado para ser escalado.

En este punto se considera **TensorFlow** [3], una librería *opensource* de aprendizaje máquina, diseñada para aprovechar las características de los grafos computacionales. Según Chris Olah, hay casos en los que implementar una red neuronal usando esta técnica mejora el rendimiento hasta 10 millones de veces [31]. *TensorFlow* provee de una API de alto nivel que facilita la creación de redes neuronales: **Keras** [9]. Dicha API es la que se ha utilizado en la elaboración de este trabajo.

3.3.2. Interfaz Gráfica

Para la interfaz gráfica se ha utilizado *Dash Open Source*, un framework que permite construir aplicaciones web interactivas de forma sencilla. La figura 9 muestra cómo se ve parte de la aplicación.

Figura 9: Vista parcial de la aplicación interactiva creada con *Dash Open Source*. El código se puede encontrar aquí.

4. Resultados

4.1. Primeros Resultados

La primera ejecución se realizó con 15 *epochs* (un *epoch* es una iteración completa sobre el conjunto de datos). Se puede ver en las gráficas cómo la red mejora su precisión en el conjunto de datos de entrenamiento, al mismo tiempo que disminuye el coste. A priori, todo parece ir según lo previsto, con una precisión del 81,89 %; sin embargo, al comprobar los datos de validación y compararlos con los de entrenamiento, se puede ver claramente que la red sufre de *sobreajuste* (*overfitting*) (figuras 10 y 11).

Uno de los objetivos principales del aprendizaje máquina es que los modelos creados sean capaces de generalizar su conocimiento. Esto es, el modelo debe ser capaz de realizar predicciones con una precisión aceptable sobre datos que nunca haya procesado.Cabría preguntarse, entonces, si un modelo bien ajustado a los datos de entrenamiento generalizará bien sobre unos datos de evaluación, siendo estos últimos desconocidos para dicho modelo.

Sabiendo que la red trata de encontrar una curva que se ajuste a los puntos, un modelo cuya curva se ciña al mayor número de datos posible será más preciso, pero la probabilidad de sobreajuste aumentará debido a que la flexibilidad de la curva puede ser demasiado alta. Esto significa que en un modelo sobreajustado, la función entre los puntos será demasiado curva, tanto que las predicciones para los nuevos valores tendrán un error mayor que el de una curva menos ajustada [39]. En otras palabras, la función de la

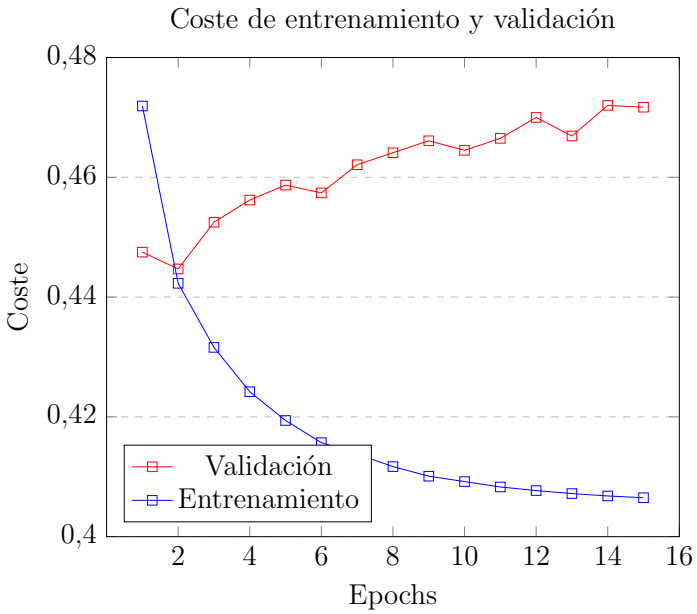

Figura 10: Gráficas de los valores de coste para los conjuntos de datos de entrenamiento y validación.

red se está acomodando a los datos y al ruido contenido en los mismos.

El sobreajuste se puede identificar fácilmente observando el punto a partir del cual la precisión de entrenamiento sube mientras que la de validación baja. De forma análoga, el coste de entrenamiento decrece mientras que el de validación aumenta. Ambas situaciones se dan en las gráficas presentadas anteriormente, a partir del segundo *epoch* (figuras 10 y 11).

Existen distintas maneras de tratar con el sobreajuste:

1. *Reducir la complejidad del modelo:* reduciendo el tamaño de las capas, o eliminándolas directamente, la capacidad de memorización de la red se reduce también.

2. *Regularización:* que consiste en minimizar la complejidad del modelo a la par que el coste.

3. *Interrupción temprana:* interrumpiendo el entrenamiento de la red en el punto de inflexión en el que se produce el sobreajuste.

Por simplicidad, se han aplicado las técnicas de reducción de la complejidad y de interrupción temprana.

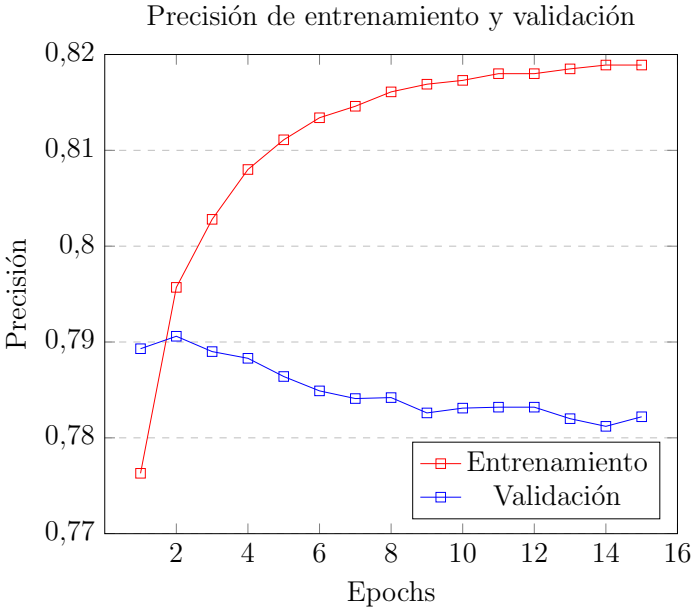

Figura 11: Gráficas de los valores de precisión de la red ejecutando el conjunto de datos de entrenamiento y validación.

4.2. Interrupción temprana

Con esta técnica se asume que, con la configuración utilizada, la red no puede tener un rendimiento mejor del que se produce en el menor mínimo de coste. En este caso, se ha realizado el entrenamiento con 3 epochs, ya que el punto de inflexión en el primer entrenamiento se produce en el epoch número 2.

Se ha decidido parar en 3 epochs en lugar de 2 para apreciar el punto de inflexión en la gráfica (figura 12), además de porque la diferencia en la precisión es mínima, permitiendo realizar tal acción sin penalizar en exceso el rendimiento de la red.

4.3. Reducción de complejidad

Hay veces que el modelo resulta demasiado complejo para la aplicación y es necesario reducirlo. En este caso, se ha realizado una reducción de la complejidad eliminando la capa estándar que se sitúa entre la capa LSTM y la capa de salida.

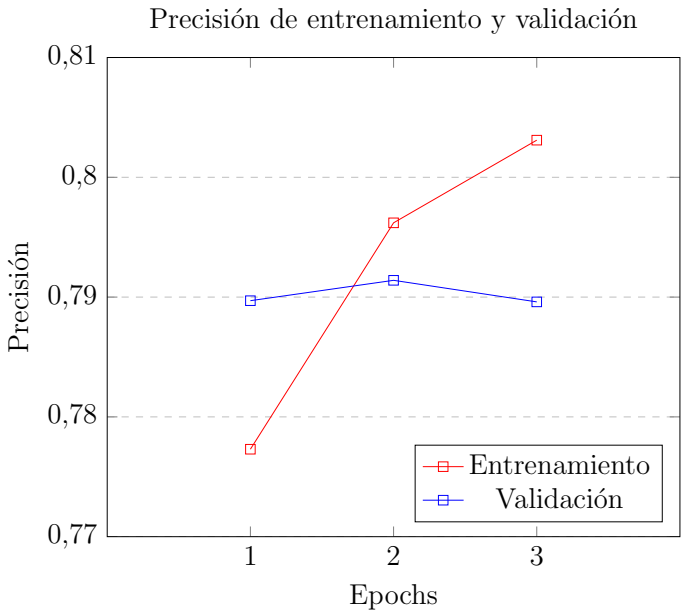

Figura 12: Gráficas de los valores de precisión de la red ejecutando el conjunto de datos de entrenamiento y validación, aplicando interrupción temprana.

Se puede apreciar en el figura 13 que reducir el modelo no tiene un gran impacto en el rendimiento de la red. Con sólo 4 epochs, ya se puede percibir el sobreajuste en el epoch número 2, donde la precisión de validación empieza a decaer.

4.4. Uso de vectores de palabras

La última iteración del proceso de creación de la red consiste en utilizar vectores de palabras predefinidos. Utilizar una representación atómica, como la codificación *one hot*, es útil y sencillo, pero no proporciona información del contexto en el que las palabras son usadas. Esto es debido a que los vectores que representan las palabras son ortogonales entre sí y, al multiplicarlos, el resultado es un vector de ceros, provocando situaciones en las que las palabras *democracia* y *gato* tienen el mismo valor estadístico. Como última desventaja, representar las palabras de forma atómica produce que los datos estén más dispersos; generalmente, esto significa que hace falta más datos para entrenar los modelos [3].

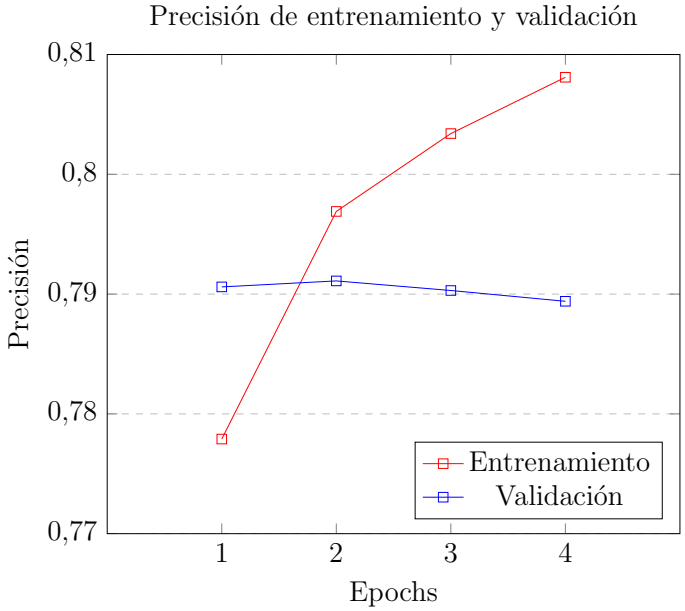

Figura 13: Gráficas de los valores de precisión de la red ejecutando el conjunto de datos de entrenamiento y validación, aplicando reducción de complejidad

Los vectores de palabras son *modelos de espacio vectorial*, y su función consiste en representar palabras en un espacio vectorial continuo, donde las palabras semánticamente similares son mapeadas con vectores también similares. Tanto los modelos *GloVe* [35] como *word2Vec* [28] han sido usados en este trabajo.

El modelo GloVe se ha utilizado para crear una matriz de pesos preentrenados, sobre la *capa embedding* de la red, que mapea las palabras a una representación vectorial continua. El modelo *word2vec* se ha utilizado para crear un espacio vectorial continuo, alimentado por las palabras contenidas en el conjunto de datos, lo que permite representar las palabras reduciendo su dimensionalidad mediante la técnica *t-SNE* [22].

Reducir la dimensionalidad de los vectores que representan las palabras abre nuevas posibilidades, como la aplicación de algoritmos no supervisados para encontrar *clusters* sobre los datos [24]. Esta técnica no sólo es útil para representar vectores de palabras, sino también para imágenes, audio y otros tipos de datos.

En las figuras 14 y 15 se puede apreciar que el sobreajuste ha desapareci-

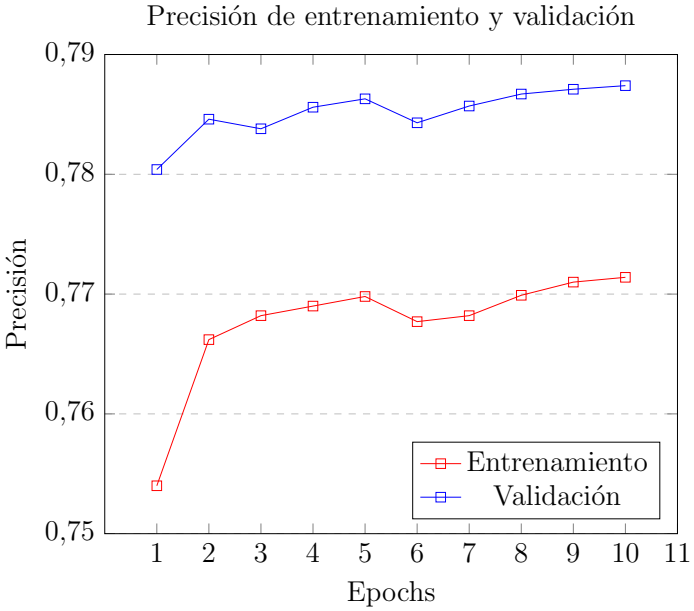

Figura 14: Gráficas de los valores de precisión de la red ejecutando el conjunto de datos de entrenamiento y validación, usando GloVe para mapear las palabras en vectores.

do, y que la red podría entrenarse más. Además, los tiempos de ejecución de un epoch usando vectores de palabras son considerablemente menores comparados con la codificación atómica: 50 minutos frente a casi 12 horas por epoch.

4.5. Evaluación mediante el coeficiente Kappa

Para evaluar el rendimiento de la red se ha utilizado el coeficienta Kappa de Cohen [11]. Este coeficiente es una medida estadística del acuerdo que tiene en cuenta la concordancia ocurrida de forma azarosa, siendo mejor que el simple cálculo del porcentaje de aciertos y fallos con respecto al total. En las publicaciones [41] y [8] se sugiere el uso del coeficiente Kappa para evaluar clasificadores, siendo el resultado satisfactorio, ya que el uso del coeficiente Kappa facilita la obtención de clasificadores más precisos.

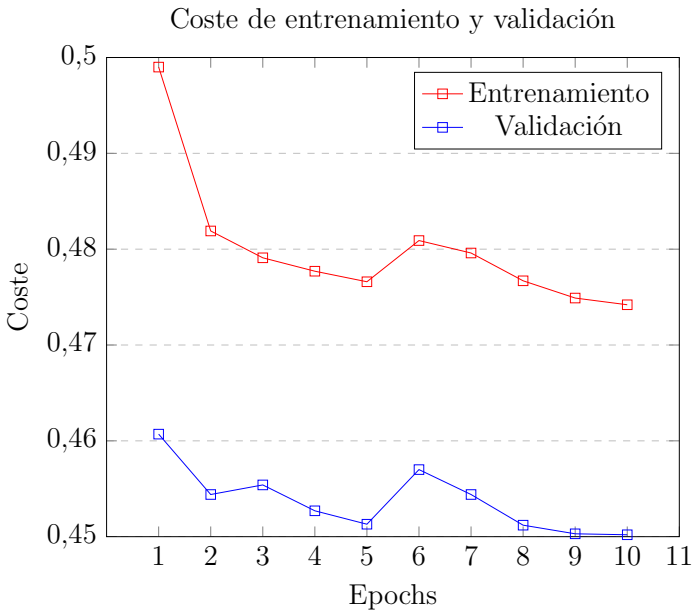

Figura 15: Gráficas de los valores de coste de la red ejecutando el conjunto de datos de entrenamiento y validación, usando GloVe para mapear las palabras en vectores.

El coeficiente Kappa de Cohen se define como:

$$k = \frac{p_o - p_e}{1 - p_e} \tag{32}$$

donde p_o es la probabilidad empírica de acuerdo sobre la clase asignada a una muestra cualquiera y p_e es la probabilidad hipotética de acuerdo por azar.

La red se evaluará sobre un conjunto de datos de 359 entradas, llamado *test*. Este conjunto viene incluido en el archivo que contiene los datos de entrenamiento, y su función no es otra que ser usado para evaluar el comportamiento final del modelo.

Dado que tanto en la creación de la red como en la transformación del texto a vectores continuos hay variables aleatorias, el coeficiente se calculará cinco veces y el resultado final será la media. Las variables que se deben tener en cuenta para calcular este coeficiente son:

1. El número de positivos reales que son evaluados como positivos. Esta variable recibe el nombre de *pp*.

	Ej. 1	Ej. 2	Ej. 3	Ej. 4	Ej. 5
pp	133	141	137	128	137
nn	135	120	122	134	131
np	49	41	45	54	45
pn	42	57	55	43	46

Cuadro 6: Tabla con los resultados obtenidos en 5 ejecuciones del modelo para el conjunto de datos *test*.

2. El número de negativos reales que son evaluados como negativos. Esta variable recibe el nombre de *nn*.

3. El número de negativos reales que son evaluados como positivos. Esta variable recibe el nombre de *np*.

4. El número de positivos reales que son evaluados como negativos. Esta variable recibe el nombre de *pn*.

5. El número total de evaluaciones. En este caso, 359.

En el cuadro 6 se pueden encontrar los valores de estas variables:

A continuación, se realizará como ejemplo el cálculo del coeficiente Kappa de Cohen para la primera ejecución del modelo. Tras este cálculo, se asume que el proceso es el mismo para las otras ejecuciones.

$$total = pp + nn + np + pn \tag{33}$$

$$p_o = \frac{pp + nn}{total} = \frac{268}{359} = 0{,}746518 \tag{34}$$

$$p_{positivo} = \frac{pp + pn}{total} \cdot \frac{pp + np}{total} = \frac{175}{359} \cdot \frac{182}{359} = 0{,}24712 \tag{35}$$

$$p_{negativo} = \frac{np + nn}{total} \cdot \frac{pn + nn}{total} = \frac{184}{359} \cdot \frac{177}{359} = 0{,}24712 \tag{36}$$

$$p_e = p_{positivo} + p_{negativo} = 0{,}49982 \tag{37}$$

$$k_1 = \frac{0{,}746518 - 0{,}49982}{1 - 0{,}49982} = 0{,}49321 \tag{38}$$

El resto de coeficientes son:

$$k_2 = 0{,}45325;\ k_3 = 0{,}44235;\ k_4 = 0{,}45996;\ k_5 = 0{,}49289 \tag{39}$$

Por tanto, la media de los coeficientes es $\bar{k} = 0{,}46833$.

La publicación [21] contiene una tabla con la *fuerza de concordancia* asociada al valor del coeficiente Kappa de Cohen. Aunque los rangos elegidos en esta tabla sean arbitrarios, proveen de una suerte de *benchmark* sobre el rendimiento. El valor obtenido en las tareas de clasificación de la red es *moderado*, ya que se encuentra en el rango $0{,}41 - 0{,}60$.

5. Conclusiones

5.1. Conclusiones sobre los resultados

Puede parecer que el coeficiente Kappa ha arrojado un resultado algo decepcionante, pero hay que tener en cuenta que dicho coeficiente puede tomar valores negativos. Además, usando la definición ingenua de probabilidad

$$P(A) = \frac{casos\ favorables\ a\ A}{casos\ posibles} \tag{40}$$

la precisión de la red sobre el conjunto de datos *test* es superior al 70 %. La calificación recibida responde a una partición en rangos arbitraria creada por un académico, con el fin de categorizar el rendimiento de un modelo de clasificación. Sin desmerecer el trabajo de otros, el modelo tiene una precisión más que aceptable.

Por último, cabe destacar que la transformación de las palabras a vectores continuos usando *GloVe* y *word2vec* ha sido clave para mantener la precisión al mismo tiempo que se reducía de forma excepcional los tiempos de ejecución de los epochs durante el proceso de entrenamiento.

5.2. Conclusiones finales

Hay mucha literatura entorno al análisis de sentimientos, pero es un campo que está en sus inicios. Parece bastante claro que las compañías valoran

muy positivamente las herramientas capaces de aplicar técnicas del procesamiento del lenguaje natural para extraer opiniones [6], pues les permite adaptar sus procesos de negocio de forma más eficaz para ahorrar costes y ganar más dinero. Los grandes agentes del mercado no son una excepción. *Google Brain*, junto con la universidad de Carnegie Mellon, dio a conocer una nueva arquitectura basada en las redes LSTM este mismo año: *Transformers XL* [12]. Esta arquitectura tiene una mayor capacidad para conectar dependencias de datos que las redes LSTM, y es de suponer que *Alphabet*, la empresa propietaria, hará uso de esta tecnología en sus servicios.

Este contexto es beneficioso para el desarrollo teórico de nuevos conceptos, arquitecturas, modelos y procedimientos en el campo del procesamiento del lenguaje natural, pues no sólo las grandes corporaciones se benefician de estos avances, aunque sí son los que disponen de más recursos para implementarlos en sus servicios y/o productos. Parece claro que el crecimiento de este campo en los próximos años está asegurado.

Referencias

[1] 7 types of neural network activation functions: How to choose? `https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/`. Accessed: 2019-5-10.

[2] Natural language processing with deep learning (winter 2017). `https://www.youtube.com/playlist?list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6`. Accessed: 2018-11-22.

[3] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[4] Yaser Abu-Mostafa. Caltech cs 156 learning from data. `http://work.caltech.edu/telecourse.html`, 2012. Accessed: 2019-05-5.

[5] Mohammad Al-Smadi, Bashar Talafha, Mahmoud Al-Ayyoub, and Yaser Jararweh. Using long short-term memory deep neural networks for aspect-based sentiment analysis of arabic reviews. *International Journal of Machine Learning and Cybernetics*, pages 1–13, 2018.

[6] D Alessia, Fernando Ferri, Patrizia Grifoni, and Tiziana Guzzo. Approaches, tools and applications for sentiment analysis implementation. *International Journal of Computer Applications*, 125(3), 2015.

[7] Yoshua Bengio, Patrice Simard, Paolo Frasconi, et al. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[8] Jean Carletta. Assessing agreement on classification tasks: the kappa statistic. *Computational linguistics*, 22(2):249–254, 1996.

[9] François Chollet et al. Keras. `https://keras.io`, 2015.

[10] Kurt Kohlstedt Christophe Haubursin, Roman Mars. The world is poorly designed. but copying nature helps. `https://www.youtube.com/watch?v=iMtXqTmfta0`. Accessed: 2019-06-4.

[11] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[12] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[13] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[14] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[15] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford*, 1(12):2009, 2009.

[16] Simon S Haykin, Simon S Haykin, Simon S Haykin, Kanada Elektroingenieur, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson education Upper Saddle River, 2009.

[17] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, Jürgen Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[19] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[20] S Sachin Kumar, M Anand Kumar, and KP Soman. Sentiment analysis of tweets in malayalam using long short-term memory units and convolutional neural nets. In *International Conference on Mining Intelligence and Knowledge Exploration*, pages 320–334. Springer, 2017.

[21] J Richard Landis and Gary G Koch. The measurement of observer agreement for categorical data. *biometrics*, pages 159–174, 1977.

[22] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[23] Mika V Mäntylä, Daniel Graziotin, and Miikka Kuutila. The evolution of sentiment analysis—a review of research topics, venues, and top cited papers. *Computer Science Review*, 27:16–32, 2018.

[24] Fernanda Viégas Martin Wattenberg, Daniel Smilkov. A.i. experiments: Visualizing high-dimensional space. `https://www.youtube.com/watch?v=wvsE8jm1GzE`, 2016. Accessed: 2019-05-27.

[25] James L McClelland, David E Rumelhart, PDP Research Group, et al. Parallel distributed processing. *Explorations in the Microstructure of Cognition*, 2:216–271, 1986.

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

[27] Fenna Miedema. Sentiment analysis with long short-term memory networks. *Research Paper Business Analytics Vrije Universiteit Amsterdam*, 2018.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[29] Andrew NG. Machine learning. `https://www.coursera.org/learn/machine-learning`, 2014. Accessed: 2018-11-10.

[30] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[31] Christopher Olah. Calculus on computational graphs: Backpropagation. `https://colah.github.io/posts/2015-08-Backprop/`, 2015. Accessed: 2019-03-30.

[32] Christopher Olah. Understanding lstm networks. `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`, 2015. Accessed: 2019-03-30.

[33] G. Orwell. *1984*. HMH Books, 1983.

[34] AB Pawar, MA Jawale, and DN Kyatanavar. Fundamentals of sentiment analysis: concepts and methodology. In *Sentiment Analysis and Ontology Engineering*, pages 25–48. Springer, 2016.

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

[36] Tariq Rashid. *Make your own neural network*. CreateSpace Independent Publishing Platform, 2016.

[37] Raúl Rojas. *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[38] Nicholas Le Roux. Using gradient descent for optimization and learning, May 2009.

[39] Open Data Science. What overfitting is and how to fix it. `https://medium.com/predict/what-overfitting-is-and-how-to-fix-it-887da4bf2cba`, 2018. Accessed: 2019-05-16.

[40] Stanford University. For academics. sentiment 140. `http://help.sentiment140.com/for-students`, 2009. Accessed: 2018-11-30.

[41] Susana M Vieira, Uzay Kaymak, and Joao MC Sousa. Cohen's kappa coefficient as a performance measure for feature selection. In *International Conference on Fuzzy Systems*, pages 1–8. IEEE, 2010.