

DOBLE GRADO EN
INGENIERÍA INFORMÁTICA Y MATEMÁTICAS



UNIVERSIDAD
DE GRANADA

TRABAJO FIN DE GRADO

**Modelos generativos del lenguaje:
aspectos teóricos y prácticos**

Autor: Alberto Estepa Fernández

Tutores: Juan Gómez Romero y Miguel Molina Solana

18 de Noviembre de 2020



E.T.S. Ingenierías
Informática y de Telecomunicación



Facultad de Ciencias

Modelos generativos del lenguaje: aspectos teóricos y prácticos

Alberto Estepa Fernández

Palabras clave: Procesamiento del Lenguaje Natural, Transformadores, Redes neuronales, Métodos de optimización de funciones, Funciones de pérdida

Resumen

En este proyecto se hace un estudio de los distintos modelos generativos del lenguaje y de la base matemática sobre la que se sustentan. Estos modelos forman la componente final de un conjunto de herramientas aptas para la resolución de tareas del campo del Procesamiento del Lenguaje Natural. Como ejemplo práctico de su uso y alcance, se incluyen dos ejemplos experimentales para generar discursos en un estilo particular y completar oraciones de forma coherente.

Generative models of language: theoretical and practical aspects

Alberto Estepa Fernández

Keywords: Natural Language Processing, Transformers, Neural Networks, Function optimization methods, Loss functions

Extended Abstract

This project presents a study of the different generative models of language, from traditional techniques (such as N-grams models) to the most current and advanced models: BERT, GPT-2, GPT-3, etc. These models form a key component of a toolchain suitable for solving language tasks, such as machine translation, search and retrieval of information, conversational systems, text summarization, etc. Research in the development of these models is part of the area of Artificial Intelligence known as Natural Language Processing (NLP).

The study and evolution of NLP has become one of the most important and current challenges of Artificial Intelligence. The complexity lies on the requirement for computers to understand the context of each word, sentence and paragraph to overcome the imperfection of the data and its ambiguity (such as ironies, sarcasm, synonymy, and polysemy, among many other issues).

Advances in the improvement of model capabilities are largely due to the great evolution of neural networks and deep learning, as well as the large amount of information that is available and can be processed.

This work reviews neural networks, from the composition of their basic unit (the artificial neuron), through the union of these units with other different ones to form a useful structure of neurons joined together, forming a complex neural network that allows to detect patterns and characteristics of the input data by creating models that serve as predictors in a different context than the input.

A neural language model calculates the probability of occurrence of a number of words in a particular sequence. The work introduces this definition together with some of the traditional language models, explaining their limitations.

Subsequently, the language models that make use of neural techniques are explained. For these neural models to be able to process the text given as input, it needs to be represented in numerical form. Several word representations are described in the work, with a special interest on the Word2Vec representation (with its two variants CBOW and

Skip-Gram).

Recurrent neural networks (RNNs) are also presented. They form a family of neural networks that allow processing data of sequential nature, which is useful for treating text. Specifically, the document details the common structure of a recurring network, its operation and its advantages and limitations. As a counterpart to these limitations, several variants of recurrent neural networks arise: the Long Short-Term Memory (LSTM) and the Gated Recurrent Units (GRU). Their structures and limitations are also detailed.

A final refinement in the architectures is that of Transformers and the mechanisms of attention that form the basis of the state-of-the-art generative models of language. A statement of its suitability and an explanation of its architecture and operation are also included.

Several well-known generative models are then presented, including TagLM and ELMo (which introduced the representation of deeply contextualized words), GPT (the first version developed by OpenAI that improved all previously published models in a large number of Natural Language Processing tasks), BERT (which improved on GPT by introducing bidirectionality in its architecture), GPT-2 (an improved version of GPT), GPT-3 (the current state of the art and the best model so far published), and other less popular models such as XLNet, ERNIE, ERNIE 2.0 and T-NLG. The transfer learning and fine-tuning processes that allow these models to be applied in practice are also discussed.

The work continues by explaining the process of training a neural network from a more formal point of view. This requires minimizing a function, which may or may not be convex. If the function is convex and differentiable, the minimum is obtained simply by differentiating and equaling to zero. In the event that this requirement is not met, there are several methods to achieve said optimization.

In this way, the loss functions most commonly used in the training of a neural network are introduced. Similarly, a comprehensive study of optimization techniques is carried out, starting from the descending gradient to other more complex optimizers.

As the culmination of this development, the mathematical process of optimization of the parameters of a feed-forward neural network is explained —the back-propagation algorithm—, and its variant for recurrent neural networks —the back-propagation through time algorithm. These algorithms are available through differentiable programming libraries that transparently calculate these derivatives.

To end this section, the suitability of neural networks for the task of approximation of functions is demonstrated by the universal approximation theorem with its different variants.

Finally, the work links this theoretical development with two experimental examples

IV

of different tasks of Natural Language Processing: the generation of speeches, and the completion of sentences. For these experiments, we use the HuggingFace Transformers library, which contains previously trained architectures and frameworks for the correct application of Natural Language Processing tasks.

For the first experiment (generating a speech for the Queen of England), we start from a variant of the GPT-2 model provided by the HuggingFace library. The fine-tuning of the model is performed by using a corpus of speeches extracted with a custom web-scraping script from the website of the British royal house. Once adjusted, the model is ready to generate speeches that share characteristics with those delivered by the Queen of England. The results are impressive: the texts are coherent and semantically and syntactically correct.

For the second experiment, we start from a variant of the BERT model provided, again, by the HuggingFace library. This model is trained with texts in Spanish, which allows the examples to be done in this language. Remarkable results are obtained: the solutions are syntactically and semantically correct but the logic behind the text is not adequate.

All in all, this works demonstrate the potential for NLP (coupled with state-of-the-art Deep Learning architectures) to appraise and mimic the communication nuances and styles of human languages.

Agradecimientos

A mi familia, gracias a los cuáles he llegado hasta aquí siendo lo que soy. A mis amigos, por el apoyo todos estos años. A mis tutores, por el arduo trabajo, su alta disponibilidad y su cercanía.

Índice general

Resumen y palabras clave	I
Abstract and keywords	II
Agradecimientos	V
1. Introducción	5
1.1. Contexto	5
1.2. Descripción del problema	7
1.3. Motivación personal	8
1.4. Contenidos	9
1.5. Planificación	10
1.5.1. Planificación temporal	10
1.5.2. Presupuestos	13
2. Objetivos	15
3. Desarrollo informático del trabajo	17
3.1. Conceptos básicos de <i>machine learning</i>	17
3.2. Redes neuronales	19
3.2.1. Neurona artificial	20
3.2.2. Capa de neuronas	22
3.3. Deep learning	24
3.4. Conceptos básicos de los modelos del lenguaje	26
3.4.1. Notación	26
3.4.2. Modelos del lenguaje: N-gramas	26
3.5. Deep learning & NLP: Modelos del lenguaje	27
3.5.1. Representación de palabras	27
3.5.2. Redes neuronales recurrentes (RNNs)	32
3.5.3. Redes neuronales recurrentes con celdas (RNNs)	36

3.5.4. Transformers	38
3.5.5. Ejemplos de modelos de lenguaje actuales	43
3.5.6. <i>Transfer Learning</i> y <i>Fine Tuning</i>	49
4. Desarrollo matemático del trabajo	51
4.1. Notación y consideraciones previas	51
4.2. Funciones de pérdida	55
4.2.1. Funciones de pérdida para problemas de regresión	56
4.2.2. Funciones de pérdida para problemas de clasificación binaria	59
4.2.3. Funciones de pérdida para problemas de clasificación multiclas	61
4.3. Métodos de optimización	63
4.3.1. Gradiente descendente	63
4.3.2. Gradiente descendente estocástico (SGD)	65
4.3.3. Momentum	66
4.3.4. Descenso acelerado de Nesterov (NAG)	66
4.3.5. Descenso de Gradiente Adaptable (Adagrad)	67
4.3.6. Adadelta	68
4.3.7. <i>Resilient backpropagation</i> (Rprop)	69
4.3.8. RMSprop	70
4.3.9. Adam	70
4.3.10. AdaMax	71
4.3.11. Nadam	72
4.3.12. ¿Qué optimizador usar en cada caso?	73
4.4. Entrenamiento de las redes neuronales <i>feed-forward</i>	74
4.5. Entrenamiento de las redes neuronales recurrentes	80
4.6. Programación diferenciable	82
4.7. Teorema de aproximación universal	82
5. Experimentación	87
5.1. Propósito y breve introducción	87
5.2. Herramientas usadas	87
5.3. Experimento 1	88
5.3.1. Diseño experimental	88
5.3.2. Resultados	90
5.3.3. Discusión	98
5.4. Experimento 2	99
5.4.1. Diseño experimental	99
5.4.2. Resultados	100
5.4.3. Discusión	102

6. Conclusiones y trabajo futuro	103
6.1. Conclusiones	103
6.2. Trabajo futuro	105
Anexo	107
A. Discursos de la reina de Inglaterra	109
A.1. Discursos sin <i>fine-tuning</i> , primera inicialización	109
A.1.1. Discurso 1	109
A.1.2. Discurso 2	111
A.2. Discursos con <i>fine-tuning</i> , primera inicialización	113
A.2.1. Discurso 3	113
A.2.2. Discurso 4	114
A.3. Discursos sin <i>fine-tuning</i> , segunda inicialización	115
A.3.1. Discurso 5	115
A.3.2. Discurso 6	116
A.4. Discursos con <i>fine-tuning</i> , segunda inicialización	118
A.4.1. Discurso 7	118
A.4.2. Discurso 8	119
Bibliografía	120

Capítulo 1

Introducción

En este capítulo se realiza una breve introducción sobre el tema a tratar, se sitúa el estado del arte de la materia y se explica la motivación del trabajo.

1.1. Contexto

El Trabajo de Fin de Grado desarrollado se engloba dentro del área de la Inteligencia Artificial, situado en el campo del Aprendizaje Automático (*Machine Learning*), más concretamente dentro del ámbito del Procesamiento del Lenguaje Natural (PLN) o, usando su traducción en lengua inglesa, *Natural Language Processing (NLP)*, notación que se usará de aquí en adelante.

Así, NLP es el área de Ingeniería de Software (IS) e Inteligencia Artificial (IA) que gestiona los lenguajes humanos. Es la técnica computacional que representa y analiza el lenguaje automáticamente [1]. Es obvio, por tanto, que otra rama que engloba dicho tema de estudio es la ciencia de la Lingüística.

Pero, ¿para qué es necesario que las computadoras puedan comprender el lenguaje natural humano? A continuación se mencionan y desarrollan algunos ejemplos prácticos y verdaderamente útiles:

- **Traducción automática de textos.** Su objetivo es traducir un lenguaje natural a otro distinto, ya sea hablado o escrito.

Dicho campo de estudio, en la rama computacional, tuvo su origen en la década de 1940 hasta el éxito del experimento de Georgetown-IBM en 1954 que consistió en la traducción de más de cuarenta frases del ruso al inglés [2]. Sin embargo no existieron grandes avances hasta finales de los 80, cuando se construyó un modelo probabilístico de traducción automática.

En la actualidad los traductores más modernos hacen uso del aprendizaje profundo

(*deep learning*) para obtener traducciones de gran calidad [3].

- **Sistemas conversacionales.** Su objetivo es obtener una comunicación en el lenguaje natural, escrito o hablado, entre una computadora y una persona.

Dentro de esta sección se engloban tanto los *chatbots* como los sistemas de diálogo, y, como ejemplos en uso actuales están los dispositivos basados en voz tales como AmazonEcho y Google Home o asistentes personales como Siri, Cortana o Google Assistant.

El origen del estudio de éste campo de la informática tuvo lugar en los años 60 cuando se crearon los primeros agentes conversacionales, aunque no eran capaces de interpretar semánticamente las frases de los usuarios, ya que se basaban en plantillas previamente definidas por el autor.

Ya en los años 70 surgen los primeros sintetizadores del habla basados en reglas, y, no es hasta la entrada del nuevo milenio cuando se enfocaría la investigación en dicho campo en incluir el aprendizaje automático para el desarrollo y evolución de los sistemas, permitiendo la interacción real entre la máquina y el usuario.

Actualmente, el aprendizaje por refuerzo y el aprendizaje profundo se aplica para construir los sistemas conversacionales modernos. [4]

- **Búsqueda y recuperación de la información.** Se encarga de extraer información de interés de los corpus de textos. El extraer entidades como lugares, eventos, horarios, precios, etc, se convierte en una necesidad para una gran cantidad de aplicaciones como, por ejemplo, los motores de búsqueda.

El interés en la evolución de dicho campo surgió en los años 70 y en la actualidad se ha convertido en un duro desafío debido a la colosal cantidad de información que se puede manejar. Los gran avances actuales están basados en los modelos de Markov ocultos (*HMMs*) y el aprendizaje profundo [5].

De forma análoga, el exceso de información de la que se puede disponer hace verdaderamente importante la capacidad de resumir datos manteniendo intacto el significado.

Esto permite además comprender los significados emocionales de ciertos textos, permitiendo a las empresas, por ejemplo, determinar el sentimiento de comentarios en redes sociales y realizar campañas de marketing personalizadas para cada cliente.

- **Aplicaciones variadas.** Además de las anteriores, el NLP permite desarrollar aplicaciones útiles como filtros de spam en el correo electrónico, resúmenes de textos, clasificación de textos de forma automática, minería de opiniones (incluso de tendencias políticas), seguimiento de conductas radicales o criminales en redes sociales

y otras múltiples aplicaciones [6].

El estudio y evolución del NLP se ha convertido en uno de los retos más importantes y actuales de la Inteligencia Artificial. La complejidad radica en que, para poder obtener información correcta de los datos debido a la cantidad de ambigüedades (como ironías, sarcasmos, distintos significados de algunas palabras según el contexto, multitud de acepciones de una misma palabra, entre otros cientos o miles de recursos lingüísticos que posee cualquier lengua) el ordenador debe comprender el contexto de cada palabra, oración y párrafo, dando un sentido global al análisis del texto.

Como se ha mencionado anteriormente, los avances en cada una de las aplicaciones expuestas se deben en gran medida a la gran evolución de las redes neuronales y del aprendizaje profundo y de la gran cantidad de información de la que se dispone y se puede procesar. Además se puede observar una tarea común entre muchas de las aplicaciones explicadas: la generación de texto de forma automática.

Este trabajo se centrará precisamente en estudiar los distintos modelos generativos del lenguaje. Para ello se estudiarán los aspectos fundamentales del aprendizaje profundo (*deep learning*) y su combinación con las diversas técnicas de NLP para desarrollar los modelos generativos actuales. Se realizará un estudio más matemático de las funciones de pérdida usadas por dichos modelos, la optimización de dichas funciones y se demostrará el por qué una red neuronal es útil en la obtención de dichos modelos. Además se probarán diferentes modelos generativos reentrenándolos para que solucionen tareas de generación de texto tanto para la generación de discursos como para tareas de autocompletar frases.

1.2. Descripción del problema

Los modelos predictivos del lenguaje son modelos de aprendizaje automático que permiten predecir el contexto más probable para un conjunto de términos (esto es, palabras anteriores o posteriores). Este tipo de modelos tienen gran utilidad en numerosas tareas de procesamiento de lenguaje natural, como el reconocimiento de entidades, la traducción automática o la generación automática de texto. En este trabajo se estudiarán las técnicas actuales basadas en aprendizaje profundo (*deep learning*) para la creación de este tipo de modelos a partir de corpus de textos.

Por otro lado, el entrenamiento de este tipo de modelos hará uso de aprendizaje profundo (*deep learning*) que se basa en la optimización de una función de pérdida, como el error cuadrático medio, aplicando gradiente descendente. Las aproximaciones más habituales emplean variantes sencillas de esta técnica, como por ejemplo el gradiente descendente estocástico, y algoritmos para agilizar la convergencia y evitar óptimos locales, como Adam o RMSProp. Sin embargo, los modelos para procesamiento de lenguaje natural actuales

requieren funciones de pérdida más sofisticadas para comparar distribuciones de probabilidad, como la divergencia Kullback-Leibler. En este trabajo se estudiarán las funciones de pérdida empleadas en los modelos generativos, la optimización de dichas funciones y la conveniencia de usar redes neuronales para la obtención de dichos modelos.

Como culminación del proyecto, se desarrollará una experimentación donde se abordará un problema de generación de lenguaje y se analizarán distintas soluciones para su entrenamiento y optimización.

Para desarrollar este proyecto se hará uso de técnicas y conocimientos adquiridos durante toda la titulación, pero el temario que predomina es el de las siguientes asignaturas:

- Ingeniería Informática:
 - Inteligencia Artificial
 - Técnicas de los Sistemas Inteligentes
 - Aprendizaje Automático
- Matemáticas
 - Análisis matemático I
 - Análisis funcional
 - Estadística Multivariante
 - Estadística Computacional

1.3. Motivación personal

El proyecto surge altamente motivado con el fin de aplicar los conocimiento adquiridos durante toda mi formación universitaria de los últimos 5 años.

Durante este periodo he podido desarrollar la suficiente (y necesaria) formación como para llevar a cabo un proyecto de tal envergadura matemática e informática, que hubiese sido muy complicado de comprender y poner en marcha sin una cimentada base de conocimientos alcanzados durante la titulación.

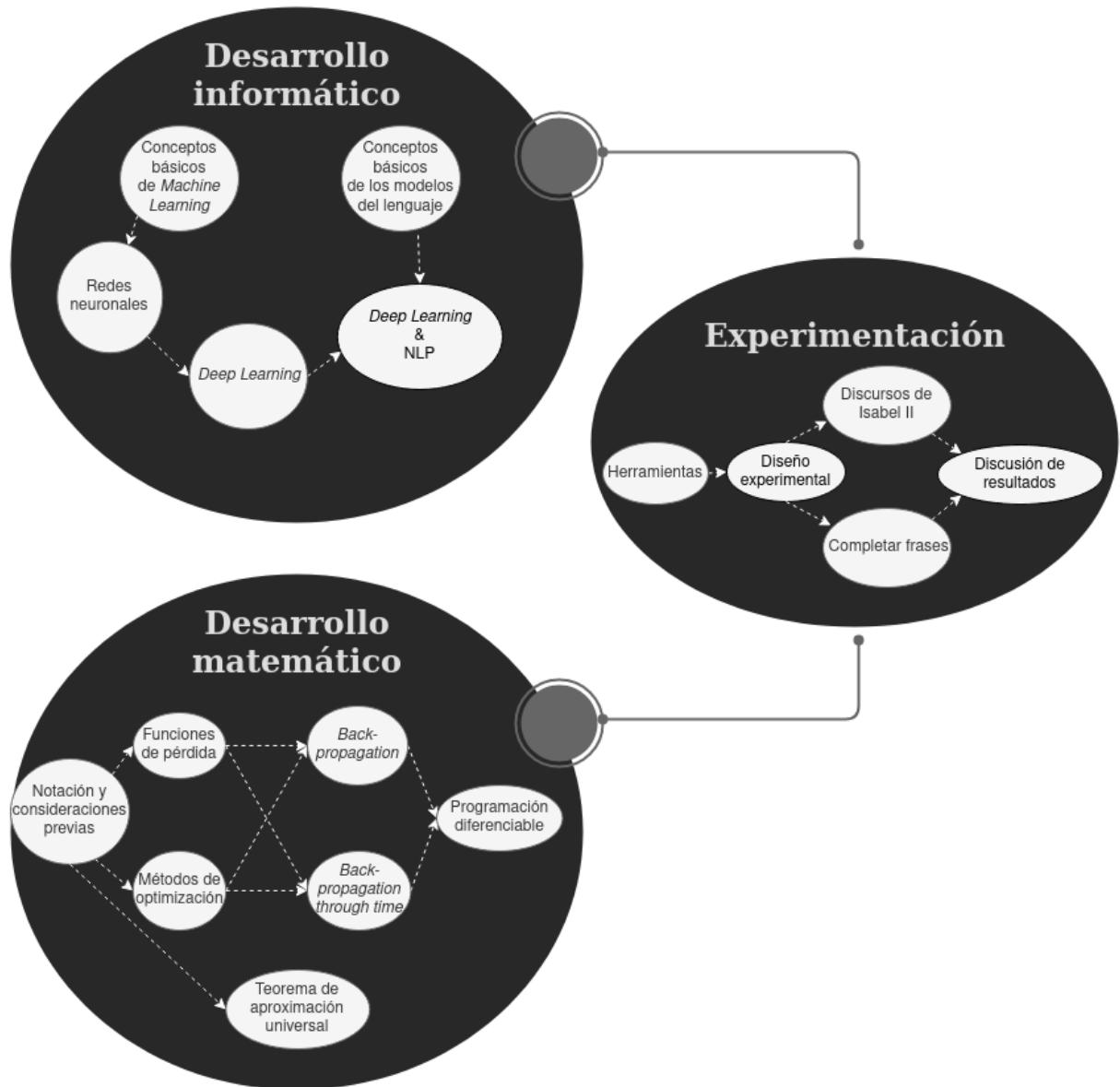
El enorme esfuerzo y dedicación que ha precisado la realización del proyecto pone en valor el nivel de formación alcanzado.

Durante el proyecto se ha llevado a cabo un estudio del área que ha experimentado el mayor crecimiento en el campo de la Inteligencia Artificial en los últimos años, convirtiéndose en la base de la revolución tecnológica en la que estamos inmersos: el *deep learning*, concretamente, en este trabajo, nos centramos en la rama de mayor auge en

estos años, el *Natural Language Processing*. Cabe destacar que durante la realización del trabajo se publicó el mayor modelo generativo hasta la fecha: GPT-3. Lo explicaremos con más detalle a lo largo de estas páginas.

1.4. Contenidos

En este trabajo se encontrarán dos ámbitos de estudio altamente relacionados que culminarán en una experimentación práctica que conecta ambos desarrollos.



Por un lado se estudiarán los conceptos básicos de *machine learning*, concretamente

la memoria se centrará en el estudio de las redes neuronales, avanzando hasta el área del *deep learning*. De forma paralela se introducirán los conceptos básicos de los modelos del lenguaje. Ambas explicaciones se combinarán para desarrollar la sección culmen de la teoría del capítulo 3. En ella se estudiarán las arquitecturas neuronales artificiales usadas a lo largo del tiempo para crear modelos de generación del lenguaje, además se introducirán los modelos de generación de lenguaje actuales y se explicará cómo usarlos en la práctica.

Por otro lado, y muy en relación con el capítulo previo, en el capítulo 4 se profundizará en los conceptos matemáticos que permitirán entrenar dichos modelos neuronales. Para ello se introducirán las principales funciones de pérdida y métodos de optimización, se explicará el proceso de aprendizaje de dichos modelos y cómo se realiza dicho proceso en la práctica, introduciendo el concepto de diferenciación automática y programación diferenciable y por último se demostrará la idoneidad de usar arquitecturas neuronales artificiales para el problema que se tiene entre manos.

Para concluir, en el capítulo 5, se aplicará la teoría estudiada en los capítulos anteriores a un problema concreto: la generación de texto. En concreto se usarán alguno de los modelos generativos del lenguajes estudiados, analizando el proceso de entrenamiento que siguen (funciones de pérdida, métodos de optimización, etc.), para generar un discurso de la reina de Inglaterra, Elizabeth II, en inglés; y para realizar de forma efectiva la tarea de autocompletar oraciones, en español.

1.5. Planificación

1.5.1. Planificación temporal

En esta sección se presenta un esquema de la planificación temporal del proyecto desde su inicio hasta la entrega del mismo. Las tareas realizadas se pueden agrupar en varias categorías:

Análisis y planificación: Esta sección incluye, como su propio nombre indica, la definición de los temas a tratar en el proyecto y la planificación a seguir. Aunque se ha desarrollado a lo largo del proyecto con reuniones de revisión periódicas, el grueso se concentra en las primeras dos semanas del ejecución del proyecto.

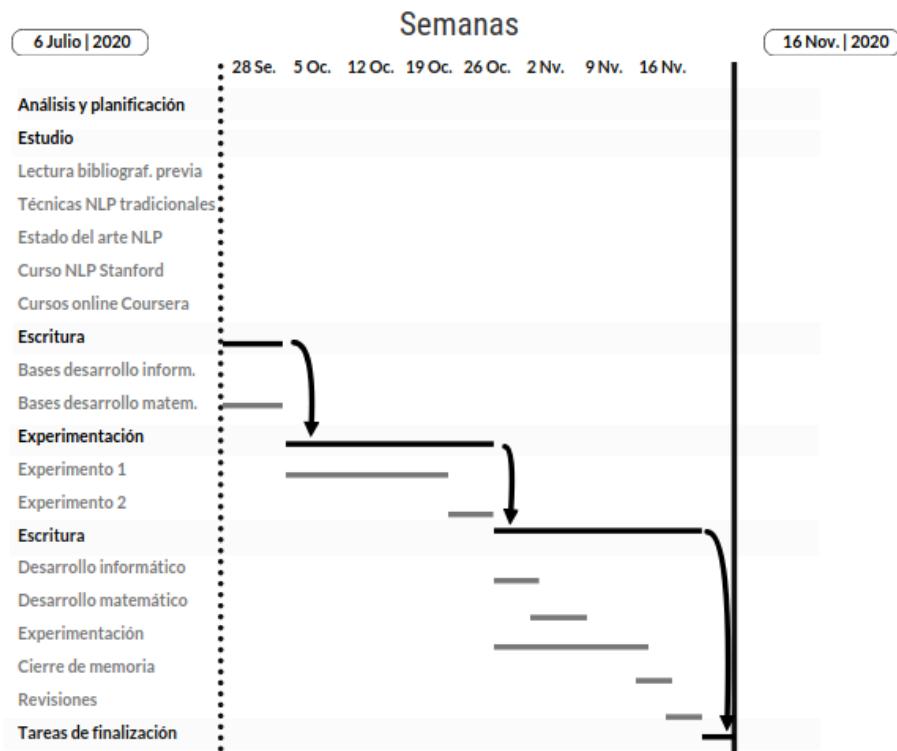
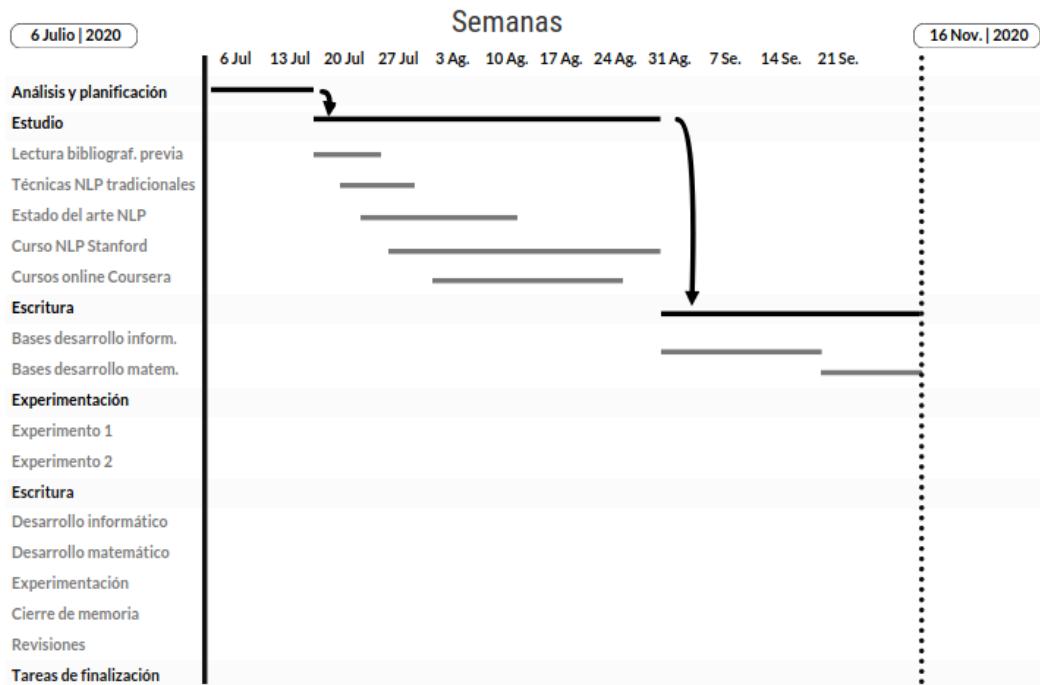
Estudio: Esta tarea comprende el estudio de los contenidos expuestos en el documento, desde la lectura de bibliografía previa incluyendo blogs científicos, artículos, etc. para obtener una visión general del estado del arte y las técnicas tradicionales. El estudio más detallado de las técnicas tradicionales y situación de contexto abarca dos semanas de ejecución. El estudio más detallado del estado del arte, incluyendo modelos actuales y arquitecturas usadas por estos abarca cuatro semanas. Paralelamente se

completaron varios cursos sobre el tema tratado, como el curso de la Universidad de Stanford *Natural Language Processing with Deep Learning* [44] o el curso disponible en Coursera *Deep Learning Specialization* [45]. La sección completa abarca un mes y medio de la ejecución del proyecto.

Escritura: Esta tarea se define cómo la escritura en L^AT_EX de este documento y se realizó durante dos períodos concretos de la ejecución del proyecto. El primer período incluye la adaptación de la plantilla, la introducción y aspectos generales de los desarrollos informático y matemático. El segundo período incluye la terminación de ambos desarrollos, la escritura y análisis de la experimentación y las revisiones y secciones que se quedaron sin terminar. En total, dicha tarea abarca 2 meses y una semana de la ejecución del proyecto.

Experimentación: Esta tarea comprende el desarrollo de los experimentos, desde la búsqueda de información hasta el análisis de resultados. Esta sección abarca un mes de ejecución del proyecto.

Tareas de finalización: Esta sección incluye la preparación del código, para que este listo y documentado para el uso general; revisiones y cierre de memoria. En total esta sección abarca una semana de la ejecución del proyecto.



1.5.2. Presupuestos

(*) Debido a que no se ha realizado un sistema software como tal, no se ha aplicado una metodología exhaustiva de cálculo de costes del proyecto.

Este proyecto se ha tasado en 450 horas de trabajo del alumno. Se estima un precio de 20 €/hora pues es lo que suele costar a una empresa un matemático o un programador junior, con costes salariales incluidos. Se incluye también el coste del tiempo de supervisión de los dos tutores del proyecto. Se estima un tiempo de 20 horas de trabajo por tutor y un coste de 40 €/hora.

Como equipo de trabajo se ha usado un portatil Lenovo ideapad 330-15IKB con un procesador Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, una memoria RAM de 8 GB y 250 GB de disco duro. Se estima su precio a 500 € a día de entrega del proyecto (18 de noviembre de 2020). Sin embargo, la vida útil de un portatil suele ser de 6 años (72 meses) y el proyecto se ha realizado en un periodo de 4 meses. Así el coste por porcentaje de uso en el proyecto equivale a 27,78 €.

Los recursos software del proyecto que se han usado son gratuitos o disponen de versión gratuita (que es por lo que se ha optado en todos los casos).

Se le añade unos gastos indirectos justificados por el mantenimiento de infraestructuras, conexión a internet, factura de luz eléctrica, etc. Estos gastos indirectos se tasan en un 5 % del total.

Presupuesto total estimado	
Precio trabajo alumno	9000 €
Precio supervisión (2 tutores)	1600 €
Costes hardware	27,78 €
Coste total (sin gastos ind.)	10.627,78 €
Gastos indirectos	531.39 €
Coste total	11.159,17 €

Precios sin incluir IVA.

Capítulo 2

Objetivos

Los objetivos fijados al inicio de la realización del proyecto se pueden agrupar en las siguientes secciones:

- Estudiar con detenimiento el área del *deep learning* para alcanzar un conocimiento suficiente y necesario con el que se fijarán las bases del desarrollo del proyecto.
- Estudiar en profundidad el proceso de evolución de los modelos generativos del lenguaje hasta la actualidad, para poder obtener una visión global de dicho campo.
- Alcanzar un alto entendimiento en las técnicas y métodos de optimización y entrenamiento de las arquitecturas neuronales artificiales que se aplican para obtener los modelos generativos del lenguaje que se van a estudiar.
- Aplicar dichos conocimientos a la práctica, realizando una experimentación sobre dichos modelos con las técnicas apropiadas y extraer conclusiones relevantes de los resultados.

El primer objetivo se trata en el capítulo 3 donde el mismo hilo conductor de dicho capítulo avanza desde las bases del *machine learning* hasta la combinación de las redes neuronales artificiales de múltiples capas junto con las bases de los modelos generativos del lenguaje (el segundo objetivo a tratar). Para mayor comprensión se realiza un estudio más matemático de dicha área en el capítulo 4, donde se demuestra la idoneidad de las redes neuronales de múltiples capas para el problema que se lleva a cabo y se explica cómo funciona al detalle el procedimiento de obtención de los modelos, obteniendo así el tercer objetivo marcado.

El segundo objetivo, como se ha mencionado en el párrafo anterior, también se incluye en el capítulo 3. Se cubre en la segunda mitad de dicho capítulo y se obtiene una visión global de los modelos generativos del lenguaje alcanzando el culmen con los más actuales hasta la fecha de realización del proyecto.

El último objetivo se cubre en el capítulo 5, donde se tratan dos problema de generación del lenguaje concretos: la generación de discursos y la completación de oraciones. Para el primer problema se harán pruebas con un corpus de discursos de la reina de Inglaterra y para el segundo se usará un de los pocos modelos previamente entrenados en español. Se explicarán las herramientas usadas para solventar dichos procedimientos y se valorarán y analizarán los resultados obtenidos.

Una vez completado el proyecto se puede afirmar que se han cumplido satisfactoriamente los objetivos propuestos al inicio.

Capítulo 3

Desarrollo informático del trabajo

3.1. Conceptos básicos de *machine learning*

Ampliando el contexto, la Inteligencia Artificial es una subdisciplina del campo de la Informática que busca la creación de máquinas que puedan imitar comportamientos inteligentes (*definición popularmente conocida desde los años 50s*). Sin embargo, se considera inteligente tanto: la acción no cognitiva, como la programación clásica que permite realizar siempre las mismas acciones; como la cognitiva, es decir, la capacidad de aprender del entorno y de la experiencia. Se profundizará en esta última vertiente.

El aprendizaje automático, *machine learning* en inglés (término que se usará de aquí en adelante), es la rama del campo de la Inteligencia Artificial que busca como dotar a las máquinas de capacidad de aprendizaje (*también popularmente conocida desde los años 50s*). Este aprendizaje consiste en identificar patrones, a partir de los datos, para realizar predicciones.

Los algoritmos de *machine learning* se clasifican en tres grandes grupos según cuál sea su paradigma de aprendizaje:

Aprendizaje supervisado: Son los algoritmos que, a partir de unos datos de entrada y su correspondiente salida, son capaces de deducir la función que establece dichas correspondencias. Al indicarle al algoritmo la salida que se quiere obtener, se “supervisa” el aprendizaje de este.

Aprendizaje no supervisado: Al contrario que en el aprendizaje supervisado, el aprendizaje no supervisado permite que los algoritmos deduzcan una función que puede producir unos datos de salida correctos solo proporcionándole datos de entrada.

Aprendizaje por refuerzo: Los algoritmos que siguen este paradigma realizan su aprendizaje recibiendo información del exterior (*feedback*) al realizar alguna acción. Esto permite al algoritmo aprender a partir de la propia experiencia o, comúnmente llamadas, acciones de prueba y error en las que se recompensan las decisiones correctas.

Pero, ¿cuáles son éstos algoritmos? Aunque existe una gran variedad, se resumen a continuación algunos de los más importantes:

- Regresión Lineal: algoritmo que permite ajustar el hiperplano que define la relación de dependencia entre las características independientes y su salida correspondiente (la variable dependiente). Pertenece al paradigma del aprendizaje supervisado. Así suponiendo N datos de entrada $\{\mathbf{x}_i, y_i\}$ con $i \in \{1, \dots, N\}$ y d la dimensión del espacio de entrada, el objetivo es encontrar la función h que minimice el error entre $h(\mathbf{x})$ (el valor de salida de la función calculada) e y (la función que representa la verdadera salida), donde:

$$h(\mathbf{x}) = \boldsymbol{\omega}^\top \mathbf{x}$$

con $h(\mathbf{x}) \in \mathbb{R}$, $\boldsymbol{\omega} \in \mathbb{R}^{d+1}$ el vector de pesos del modelo ($\boldsymbol{\omega} = [\omega_0, \omega_1, \omega_2, \dots, \omega_d]^\top$) y $\mathbf{x} \in \mathbb{R}^{d+1}$ la variable de entrada ($\mathbf{x} = [x_0, x_1, x_2, \dots, x_d]^\top$ con $x_0 = 1$) [7].

- Regresión Logística: algoritmo que permite predecir el resultado de una variable categórica en función de las variables independientes. De nuevo, con las hipótesis del modelo anterior, la función h vendrá dada esta vez por:

$$h(\mathbf{x}) = \theta(\boldsymbol{\omega}^\top \mathbf{x})$$

donde $\theta(s) = \frac{e^s}{1+e^s} \in [0, 1]$ lo que se interpreta como una probabilidad entre dos clases. Este método se puede generalizar a un número de clases de salida mayor y además se encasilla dentro del paradigma de aprendizaje supervisado [7].

- Árboles de decisión: familia de algoritmos cuyo objetivo es crear modelos que predicen la clase o el valor de la variable objetivo mediante reglas de decisión simples inferidas de los datos de entrenamiento. Sirven tanto para regresión como para clasificación y están encuadrados en el paradigma del aprendizaje supervisado, aunque pueden usarse bajo el paradigma del aprendizaje no supervisado, como para técnicas de *clustering*. Existen diversos tipos de árboles de decisión que constituyen esta familia, como: ID3, C4.5, CART, CHAID, MARS, etc. E incluso, a grandes rasgos, la combinación de múltiples árboles de decisión será la clave de los algoritmos conocidos como Random Forest y AdaBoost. No se profundizará en ellos en este trabajo, ya que no se incluye en el objetivo de éste.

- Máquinas de vectores de soporte (*Support Vector Machines, SVM*): algoritmo cuyo objetivo es encontrar un hiperplano que separe de forma óptima (es decir, que los puntos más cercanos a él se sitúen a distancia máxima) al conjunto de clases. Pertenece al paradigma del aprendizaje supervisado y aunque es comúnmente usado en problemas de clasificación, también existe una variante para problemas de regresión.
- K-medias (*K-means*): algoritmo cuyo objetivo es agrupar elementos del espacio en K clústers minimizando la suma de distancias entre el elemento del espacio y el centroide de su clúster. Pertenece al paradigma del aprendizaje no supervisado y resuelve problemas de agrupamiento.
- Naive Bayes: familia de algoritmos basados en el teorema de Bayes o probabilidad condicionada. Calculan las probabilidades condicionadas suponiendo la independencia de las características entre sí, y establecen una regla probabilística para escoger la clase resultante. Son algoritmos altamente sencillos pero muy útiles cuando se tiene una gran cantidad de datos. Son usados, por ejemplo, en clasificación de textos y están basados en el paradigma del aprendizaje supervisado.
- Redes neuronales: familia de algoritmos cuya estructura común es la unión de unidades básicas de procesamiento (llamadas neuronas artificiales) que realizan operaciones entre ellas para descubrir patrones que caracterizan y discriminan los datos de entrada obteniendo una predicción de salida. Para conseguir dicho resultado, la red pasa por un periodo de entrenamiento que consiste en intentar minimizar una función de pérdida actualizando los escalares que influyen en las operaciones realizadas en las neuronas. Este proceso se denomina *backpropagation*.

Por supuesto existen otros algoritmos (como el algoritmo de K-NN, la regresión escalonada, el algoritmo apriori, algoritmos de reducción de dimensionalidad, etc.) que no se han incluido en la lista anterior ya que esto no es objeto de estudio de este trabajo.

Retomando el hilo conductor del proyecto, como la teoría desarrollada a continuación se basa en una implementación de redes neuronales, el siguiente paso, en este trabajo, será el de estudiar dicha familia de algoritmos.

3.2. Redes neuronales

Para empezar a desarrollar la teoría de redes neuronales, lo más conveniente es estudiar cómo funciona la unidad básica de procesamiento que forma una red neuronal: la neurona artificial.

Se proseguirá con el estudio de las agrupaciones de las neuronas artificiales para formar la red neuronal y cómo estos sistemas son capaces de desarrollar su aprendizaje.

3.2.1. Neurona artificial

Una neurona artificial es la unidad básica de procesamiento de las redes neuronales. El mecanismo de funcionamiento es sencillo: la neurona recibe la información del exterior mediante datos de entrada $\mathbf{x} = [x_0, x_1, \dots, x_N]^\top \in \mathbb{R}^{N+1}$ (si se consideran N datos de entrada); con estos valores, la neurona realizará operaciones para producir una única salida $y \in \mathbb{R}$. Por tanto se está realizando una simple función matemática a los datos de entrada.

Las operaciones internas se limitan a: una suma ponderada de todos los datos de entrada y una función de activación.

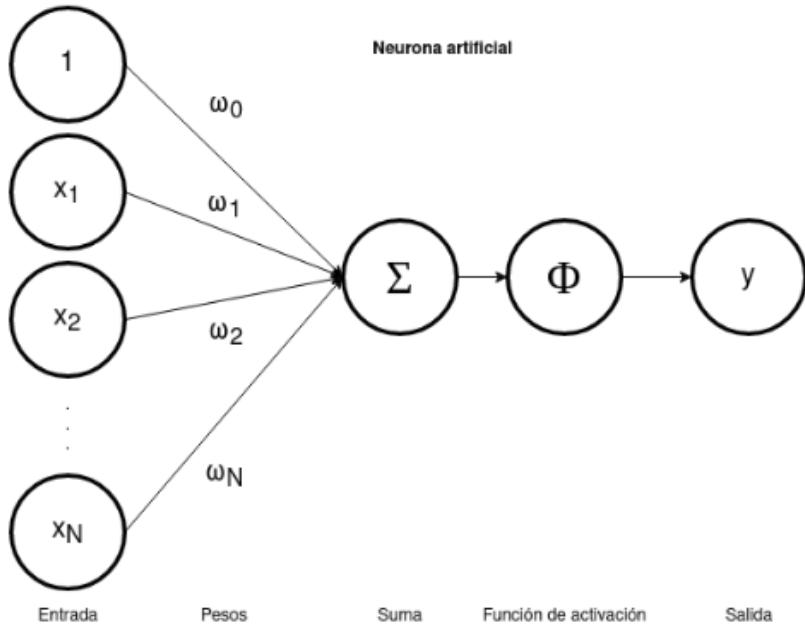


Imagen 3.1: Esquema ilustrativo de las partes de una neurona artificial

La suma ponderada consiste en la suma de todos los datos de entrada multiplicados, cada uno de ellos, por un escalar, comúnmente llamado peso ($\omega_i \in \mathbb{R}$ con $i \in \{1, 2, \dots, N\}$). Generalmente, como se ha introducido anteriormente, se suele añadir una entrada adicional, $x_0 = 1$, que lleva asociado un peso, $\omega_0 \in \mathbb{R}$, que se denomina sesgo (*bias*, en inglés) y esta asociado a la letra b ($\omega_0 = b$). Dicho peso ajusta la predisposición de la neurona para devolver una salida positiva o no independiente de las entradas. Así un sesgo alto indica que la neurona necesita unos valores de entrada más grandes para devolver una salida positiva y uno bajo permite que no se necesiten unos valores de entrada tan altos para devolver dicha salida. De esta forma, el conjunto de pesos de la neurona se notará como $\boldsymbol{\omega} = [\omega_0, \omega_1, \omega_2, \dots, \omega_N]^\top \in \mathbb{R}^{N+1}$. Los pesos representan la influencia de cada una de las entradas en la información de la neurona y serán los parámetros a ajustar para que

el modelo desarrolle su aprendizaje.

Por otro lado, la función de activación recoge el resultado de la suma ponderada anterior y le aplica una transformación no lineal. Existen diversas funciones de activación que se pueden aplicar:

- Función escalonada: Transforma el resultado de la suma ponderada en 1 o 0 dependiendo de si el valor de la suma es mayor que un umbral, que suele ser 0. Formalmente:

$$f(\omega^\top \mathbf{x}) = \begin{cases} 0 & \text{si } \omega^\top \mathbf{x} < \text{umbral} \\ 1 & \text{si } \omega^\top \mathbf{x} \geq \text{umbral} \end{cases}$$

Se visualiza dicha función en la imagen 3.2a. Sin embargo esta función realiza de forma brusca la discriminación entre las dos posibles salidas lo que hace que dicha función no sea la más apropiada para su aplicación en las redes neuronales. Las siguientes funciones de activación solucionan éste problema.

- Función sigmoide: Análoga a la función escalonada, pero derivable, evitando el cambio brusco descrito anteriormente. Formalmente:

$$\sigma(\omega^\top \mathbf{x}) = \frac{1}{1 + e^{-\omega^\top \mathbf{x}}}$$

Se visualiza dicha función en la imagen 3.2b. Fue bastante popular durante mucho tiempo en el campo del *machine learning*, pero actualmente está en desuso ya que en las redes neuronales profundas se complica el entrenamiento debido al problema del desvanecimiento del gradiente (*vanishing gradient problem*, en inglés) en el que, resumiendo, el gradiente, que será la clave en la etapa de aprendizaje de la red, se vuelve insignificante en las primeras capas, lo que dificulta el entrenamiento (dicho problema se tratará con más detalle en la sección 3.3).

- Función unidad lineal rectificada (ReLU): Función actualmente muy utilizada que evita la saturación en los extremos que provocaba la función sigmoide. Es una función constante a cero para valores negativos y lineal con respecto a la entrada para los valores positivos. Formalmente:

$$R(\omega^\top \mathbf{x}) = \max(0, \omega^\top \mathbf{x}) = \begin{cases} 0 & \text{si } \omega^\top \mathbf{x} < 0 \\ \omega^\top \mathbf{x} & \text{si } \omega^\top \mathbf{x} \geq 0 \end{cases}$$

Se visualiza dicha función en la imagen 3.2c. Es importante señalar que aunque no exista la derivada en 0, computacionalmente se suele extender con $f'(0) = 0$.

- Función softmax: Función que suele usarse en la última capa de la red. Proporciona como resultado un vector de números reales no negativos que suman uno, lo que lo convierte en una distribución de probabilidad discreta sobre los resultados posibles. Se utiliza cuando se desea modelar una distribución de probabilidad sobre las posibles clases de salida ($\{1, \dots, k\}$).

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}}$$

- Existe una gran variedad de funciones de activación que presentan ventajas o inconvenientes según el tipo de red o la capa de ésta que contenga a la neurona. Otras funciones de activación son la función tangente hiperbólica, la función arcotangente, la función SoftSign, etc., y otras variantes de las anteriormente expuestas.

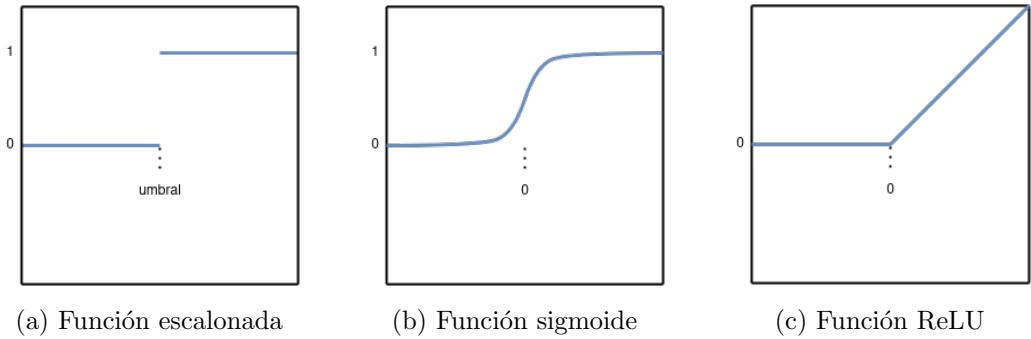


Imagen 3.2: Visualización de las funciones de activación escalonada, sigmoide y ReLU.

Así, en resumen, una neurona artificial recibe la información con un conjunto de entradas ($\mathbf{x} = [1, x_1, \dots, x_N]^\top \in \mathbb{R}^{N+1}$) y mediante una suma ponderada de éstas, seguida de la aplicación de una función no lineal (la función de activación correspondiente, $\phi : \mathbb{R} \rightarrow \mathbb{R}$) obtiene una salida:

$$y = \phi \left(\omega_0 + \sum_{i=1}^N x_i \omega_i \right) = \phi(\boldsymbol{\omega}^\top \mathbf{x})$$

donde $\boldsymbol{\omega} = [\omega_0, \omega_1, \omega_2, \dots, \omega_N]^\top \in \mathbb{R}^{N+1}$ son los pesos de la neurona.

3.2.2. Capa de neuronas

Extendiendo la idea anterior a múltiples neuronas, es decir, considerando que los datos de entrada se proporcionan a varias neuronas a la vez, se obtiene una capa de una red neuronal (Figura 3.3).

Suponiendo m neuronas en la capa de estudio, los pesos de las neuronas que la forman se denotan con

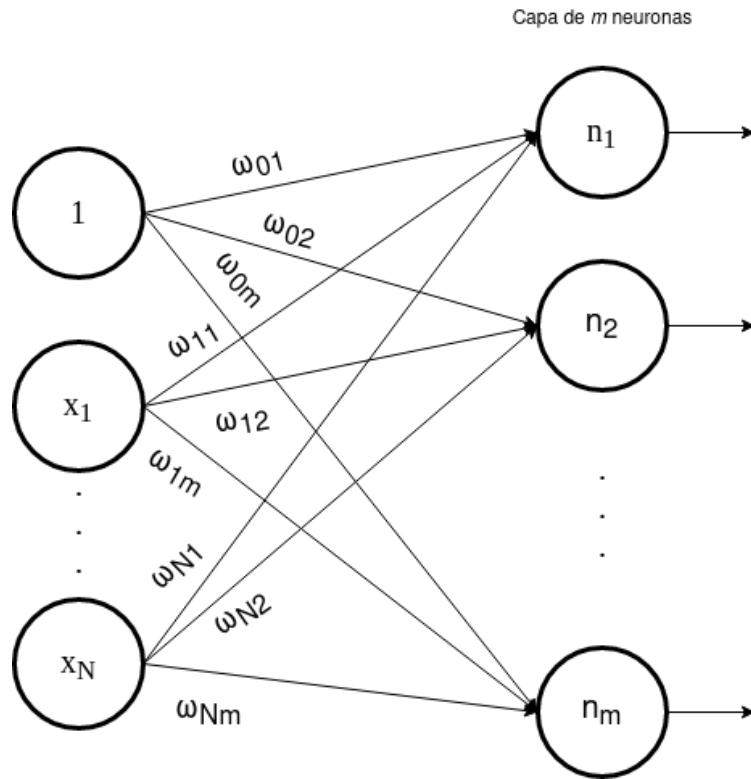


Imagen 3.3: Esquema ilustrativo de una capa de una red neuronal.

$$\omega_1 = [\omega_{01}, \omega_{11}, \omega_{21}, \dots, \omega_{N1}]^\top \in \mathbb{R}^{N+1}$$

$$\omega_2 = [\omega_{02}, \omega_{12}, \omega_{22}, \dots, \omega_{N2}]^\top \in \mathbb{R}^{N+1}$$

⋮

$$\omega_m = [\omega_{0m}, \omega_{1m}, \omega_{2m}, \dots, \omega_{Nm}]^\top \in \mathbb{R}^{N+1}$$

formando así una matriz $\mathbf{W} = [\omega_1^\top, \omega_2^\top, \dots, \omega_m^\top] \in R^{m \times (N+1)}$ de pesos de la capa.

Por tanto, la salida $\mathbf{y} \in \mathbb{R}^m$ proporcionada por la capa de neuronas se obtiene como:

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x})$$

donde $\phi : \mathbb{R}^m \rightarrow \mathbb{R}^m$ es una función que aplica cada función de activación de cada neurona a la suma ponderada obtenida en su correspondiente neurona.

El procesamiento anterior ha dado como resultado una salida en forma de vector con tantas componentes como neuronas forman la capa que ha procesado los datos. Los valores de salida de esta capa suelen servir como entrada de una capa de neuronas denominada

“capa de salida”, cuyo objetivo es procesar la información de la anterior capa a una salida esperable, por ejemplo, si se está ante un problema de clasificación en tres grupos, lo habitual es disponer de una “capa de salida” de tres neuronas cuya función de activación, de las neuronas de esta capa, sea la función *softmax*, que permite asignar probabilidades a cada grupo o clase.

Por definición, la entrada de los datos se considera una capa, aunque no se realice un procesamiento de los datos, y se denomina “capa de entrada”. La capa que se encuentra entre la capa de entrada y la capa de salida se denomina “capa oculta”. En el contexto del ejemplo anterior, lo que se ha implementado es una red neuronal de una capa oculta.

De esta forma, se ha definido la estructura de una red neuronal sencilla, un proceso computacional que permite modelar comportamientos inteligentes. Se denomina a estas redes redes neuronales *feed-forward*, aunque como son las más básicas se entiende que al mencionar el término general “redes neuronales”, se refiere a dichas redes.

A pesar de la simplicidad de la red descrita, en la práctica, lo habitual es necesitar varias capas ocultas, lo que proporcionará poder para aprender conocimiento jerarquizado, o información cada vez más elaborada. Así la salida de la primera capa oculta podrá servir como entrada de una siguiente capa oculta, y así sucesivamente hasta llegar a la capa de salida. Esta profundidad en la cantidad de capas es lo que da nombre al Aprendizaje Profundo, o *deep learning* en inglés (notación que se usará a partir de ahora).

3.3. Deep learning

El *deep learning* es un área del *machine learning* cuya base son las redes neuronales con múltiples capas ocultas. Como ya se ha mencionado, las redes neuronales permiten desarrollar un conocimiento de forma jerarquizada aprendiendo por niveles (bloques y capas de la red) diferentes patrones o características, donde, en los primeros niveles, se distinguen patrones sencillos y, en los niveles posteriores, se usan dichas características para detectar patrones más elaborados.

Es importante destacar, en este punto, el valor de la función de activación de las neuronas en una red neuronal, y se pone en valor con la profundidad de la red: si estas funciones no existiesen, la red solo produciría decisiones lineales, ya que la combinación lineal de funciones lineales se resume en una simple función lineal, es decir, ya podría ser una red muy profunda que el mismo resultado podría conseguirse simplificando la estructura de la red en una simple neurona. Por tanto, la no-linealidad que incluyen las funciones de activación permiten aproximar funciones complejas, motivo por el cual una red neuronal es una herramienta de cómputo muy valiosa. La idoneidad de dichas arquitecturas para la aproximación de funciones es demostrada por el teorema de aproximación universal

que se detallará en la sección 4.7.

Idealmente, el conocimiento que se puede desarrollar por una red avanza a medida que se incluyen más niveles en nuestro modelo de red neuronal. Pero, en la práctica, existen ciertos problemas que limitan o hacen poco efectivo la acumulación de capas, aunque solventando estos problemas, la tendencia en la actualidad es la de incluir cada vez más y más capas.

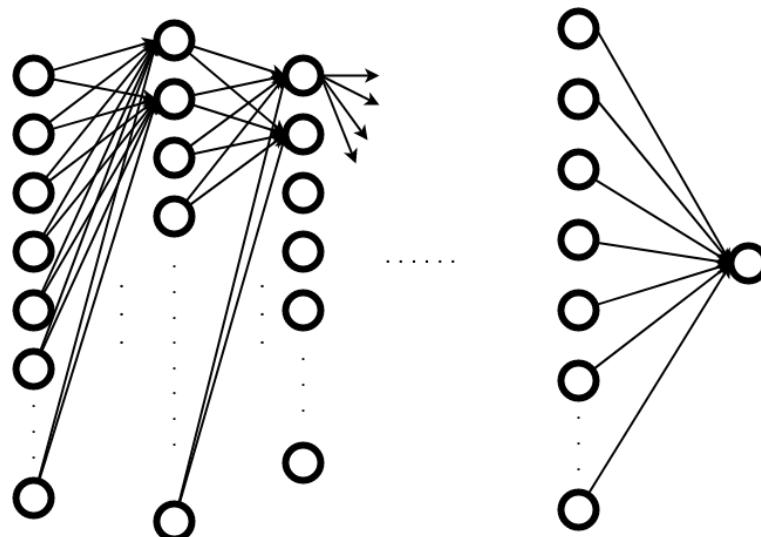


Imagen 3.4: Esquema ilustrativo de una red neuronal con múltiples capas ocultas.

Hasta ahora se ha construido una herramienta con la que variando sus parámetros se pueden resolver problemas complejos de clasificación o regresión. Sin embargo no se ha mencionado cómo es capaz una red neuronal de autoajustar los parámetros para que a partir de los datos la red desarrolle su aprendizaje. Esta técnica se conoce como *backpropagation* y dado su fundamento matemático, se explicará en el capítulo 4, más concretamente en la sección 4.4.

Pues bien, uno de los problemas que surgen en las redes profundas en la etapa de entrenamiento (concretamente cuando se realiza la retropropagación del error) es la parálisis de la red o *vanishing gradient*. Como se explica en la sección 4.4, para calcular el error imputado a una neurona se aplica un producto de tres elementos, por lo que, si alguno de estos elementos (como por ejemplo la derivada de la función de activación) tuviera un valor muy bajo, la actualización de los parámetros sería prácticamente nula. Por esto es recomendable usar funciones de activación como la función ReLU (explicada en la sección 3.2.1).

Otras técnicas que mejoran el rendimiento de una red neuronal profunda son: la normalización de los datos, tanto en la entrada de la red como durante el entrenamiento [12], y técnicas como el *dropout* [13] y la regularización [14] que mitigan los efectos del *overfitting*.

y del *underfitting*. No se profundizará en dichas técnicas ya que no es objeto de estudio en este trabajo.

Una vez explicada, de manera resumida, la base teórica de la herramienta que va a permitir desarrollar los modelos objetivo en el proyecto, se procede a continuación a abordar el área del *Natural Language Processing*.

3.4. Conceptos básicos de los modelos del lenguaje

Antes de la eclosión de los sistemas de *deep learning* en el campo del *Natural Language Processing*, la generación de texto se basaba en sistemas puramente estadísticos. En esta sección se introducen las nociones básicas de los modelos del lenguaje y estos primeros modelos.

3.4.1. Notación

Los modelos del lenguaje calculan la probabilidad de ocurrencia de un número de palabras en una secuencia particular. Denotando por $x^{(i)}$ con $i \in \mathbb{N}$ a una palabra del vocabulario \mathcal{V} , la probabilidad de una secuencia de m palabras $\{x^{(1)}, \dots, x^{(m)}\}$ se denota como $P(x^{(1)}, \dots, x^{(m)})$.

Siguiendo con la notación anterior, dada la secuencia anterior de palabras, se denota con $P(x^{(m+1)}|x^{(1)}, \dots, x^{(m)})$ a la probabilidad de generar la palabra $x^{(m+1)}$ como sucesora de la secuencia $\{x^{(1)}, \dots, x^{(m)}\}$.

Se considera que $P(x^{(1)}, \dots, x^{(m)})$ está condicionada a un conjunto de palabras anteriores seguidas en vez de a todas las palabras predecesoras, ya que el número de palabras que se sitúan antes de una palabra concreta, $x^{(i)}$, varía según su ubicación en el documento de entrada. Esta afirmación se formaliza como:

$$P(x^{(1)}, \dots, x^{(m)}) = \prod_{i=1}^m P(x^{(i)}|x^{(1)}, \dots, x^{(i-1)}) \approx \prod_{i=1}^m P(x^{(i)}|x^{(i-n)}, \dots, x^{(i-1)})$$

3.4.2. Modelos del lenguaje: N-gramas

Modelo probabilístico que supone que la palabra $x^{(m+1)}$ depende solo de las $N - 1$ palabras anteriores [44]:

$$\begin{aligned} P(x^{(m+1)}|x^{(1)}, \dots, x^{(m)}) &= P(x^{(m+1)}|x^{(m-N+2)}, \dots, x^{(m)}) = \\ &= \frac{P(x^{(m+1)}, x^{(m)}, \dots, x^{(m-N+2)})}{P(x^{(m)}, \dots, x^{(m-N+2)})} \end{aligned} \tag{3.1}$$

donde, en este último paso, se ha aplicado la definición de probabilidad condicionada.

Estas probabilidades se calculan contando el número de veces que aparecen dichas secuencias de palabras en el corpus de textos de entrenamiento.

La simplicidad de dicho modelo provoca varios problemas:

- La suposición de que la palabra $x^{(m+1)}$ depende solo de las $N - 1$ palabras anteriores provoca, en la práctica, que se deseche parte del contexto de la palabra y, por tanto, se obtiene una peor predicción.
- Puede ocurrir que en el corpus de textos usado para entrenar el modelo no aparezca nunca la secuencia $\{x^{(1)}, \dots, x^{(m)}, x^{(m+1)}\}$ (numerador de la fórmula 3.1), siendo $x^{(m+1)}$ la palabra que se desea en la práctica, lo que provocaría que el modelo no generase nunca dicha secuencia de palabras. Este problema se puede solventar suavizando el modelo: añadiendo una pequeña probabilidad a cada una de las palabras del vocabulario para cada secuencia.
- De igual forma, puede ocurrir que en el corpus de textos no aparezca nunca la secuencia $\{x^{(1)}, \dots, x^{(m)}\}$ (denominador de la fórmula 3.1), por lo que no se podría calcular la probabilidad para ninguna palabra. Esto se puede solucionar disminuyendo el número de palabras que tiene en cuenta el modelo (pasar de N a $N - 1$, o incluso más), perdiendo información de contexto.

Como se ha estudiado, dicho modelo tiene bastantes limitaciones: un número elevado para el parámetro N provoca que el modelo no sea funcional, ya que el número ocurrencias en el corpus de textos de las secuencias de palabras podría ser cero; mientras que un número pequeño provoca desechar el contexto y por tanto una peor predicción.

3.5. Deep learning & NLP: Modelos del lenguaje

Una vez contextualizado y profundizado en las bases teóricas del proyecto, procedemos a indagar sobre las técnicas y los modelos del lenguaje que se han utilizado y han evolucionado a partir del auge del *deep learning* en la rama del *Natural Language Processing*. El objetivo final será conocer y profundizar sobre los modelos generativos del lenguaje más actuales.

3.5.1. Representación de palabras

Los modelos de generación del lenguaje que se van a explicar a continuación están basados en redes neuronales y como tal necesitan que la información que reciban como entrada esté codificada de forma numérica.

Existen diversas estrategias para codificar la información del lenguaje, cada una de ellas tendrá su utilidad según el problema que se presente:

- Codificación de palabras: Si se trabaja con un vocabulario de palabras ya formado, será más sencillo generar textos que parezcan más cercanos a la realidad.
- Codificación de caracteres: Generar una palabra como una secuencia de caracteres permite que los modelos puedan generar nuevas palabras: palabras que no han sido incluidas en el vocabulario o que no existen.
- Codificación de sub-palabras: Es un método alternativo, situado entre medias de los dos anteriores, que, por ejemplo, es la base de representación de métodos muy actuales como GPT-2 y GPT-3.

A cada uno de estos bloques que forman la secuencia de datos, ya sean palabras, caracteres o subpalabras, se le denomina token y al proceso de división de los datos en tokens, tokenización.

El trabajo se centrará en explicar la codificación a nivel de palabras, aunque se puede generalizar fácilmente a cualquiera de las otras dos codificaciones. De esta forma, existen varias representaciones posibles de representar cada token en el lenguaje, en este caso, cada palabra:

- A través de un número, con una relación unívoca entre dicho número y cada token. Sin embargo, esta representación proporciona a los algoritmos poca información sobre la similitud y la representación espacial entre las palabras. Por ejemplo, si se representa la palabra “ordenador” con el valor 10, “portatil” con el valor 20 y “ladrillo” con el valor 15, el modelo podrá entender que un “portatil” es el doble de un “ordenador” y que “portatil” se parece más a un “ladrillo” que a un “ordenador”, por la distancia a la que se encuentra.
- Mediante una representación ***One-Hot Encoding*** que asigna un vector distinto a cada token. Este vector será del mismo tamaño que el vocabulario (tendrá tantas posiciones como tokens distintos existen). Además, se marca con 1 solo una posición (la que corresponda al token), y con 0 las demás posiciones, lo que hace unívoca dicha representación.

De esta forma, los tokens serán vectores ortonormales. De hecho los vectores están contenidos en un espacio vectorial de dimensión $|\mathcal{V}|$ donde cada uno de ellos se sitúa a la misma distancia de todos los demás, por lo que se impide que el modelo a entrenar aprenda patrones por la similitud en los datos.

Sin embargo esta representación es bastante ineficiente en memoria y gran parte de la información (la mayoría de las componentes del vector que representa un token) es 0. Por otra parte, en un vocabulario existen palabras con mayor similitud que

otras, por ejemplo: los sinónimos y otros tipos de relaciones semánticas; lo que puede aportar información valiosa para los modelos y con esta representación no es posible obtenerla. Estos problemas se resuelven con la siguiente representación.

- Usando una representación en *embeddings* que representa los tokens con vectores de números reales. Para obtenerla se debe realizar una reducción de dimensionalidad de un espacio con una dimensión por palabra (como en *One-Hot Encoding*) a un espacio vectorial de menor dimensión. Se muestra un ejemplo proporcionado por el proyector de *embeddings* [TensorBoard](#) (imagen 3.5).

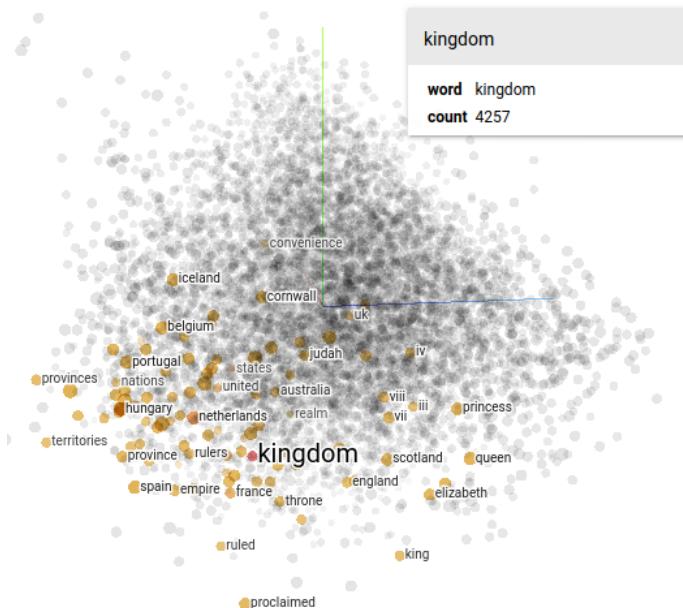


Imagen 3.5: Representación de la palabra “kingdom” con el modelo Word2Vec con un vocabulario de 10.000 palabras, en inglés, en *embeddings* de dimensión 200, una vez realizada una PCA a dimensión 3. También se señalan las representaciones en *embeddings*, en el mismo espacio de vectores, de palabras relacionadas semánticamente con “kingdom”.

Se debe tener en cuenta que la dimensión original del *embeddings* es 200 y en la imagen se visualiza en solo 3 (después de aplicar PCA), ya que el ojo humano solo puede apreciar imágenes en tres dimensiones, incluso es difícil de apreciar esta tercera dimensión. Esto provoca que mucha información original se pierda en la visualización y no sea realmente representativa la imagen, pero da una idea aproximada de cómo funciona la representación: en nuestro caso las palabras con mayor similitud a “kingdom” son las marcadas en color, como “throne” o nombres de países relacionados con reinos como “netherlands”, “france” o “england”. Se observa que todas las palabras similares se encuentran en la parte inferior y menos profunda de la imagen, cerca de “kingdom”.

La representación obtenida por parte de los *embeddings* permite que los modelos pue-

dan desarrollar información de similitud entre los tokens o palabras e incluso puede llegar a construir un espacio matemático donde se pueden implementar diferentes álgebras. El ejemplo más famoso es el dado por Word2Vec en que si se realiza la operación entre el *embeddings* que representan la palabra “King”, restada con el *embedding* que representa la palabra “Man” y sumada al *embedding* que representa la palabra “Woman”, se obtiene como resultado aproximado el *embedding* que representa la palabra “Queen”. Es decir:

$$\text{vector}(King) - \text{vector}(Man) + \text{vector}(Woman) \approx \text{vector}(Queen)$$

Para obtener la representación de los tokens en *embeddings* se puede usar la representación en *One-Hot Encoding* como entrada de una red neuronal cuya capa de salida será del tamaño de la dimensión del *embedding* que se quiera fijar. La red irá ajustando los pesos mediante las técnicas de optimización oportunas y se obtendrá la información de la entrada de forma compactada en vectores de dimensión deseada. En un comienzo la red neuronal no aportará información de similitud correcta, pero cuando empiece el entrenamiento de la red, poco a poco se comprimirá y ordenará cada uno de los *embeddings* de manera adecuada para resolver el problema con el que se ha entrenado. Los *embeddings* pueden conseguir ser representados según su semántica.

Otra alternativa será partir de una red previamente entrenada de *embeddings*, puesto que parte del conocimiento ya aprendido podrá ser transferido al nuevo problema (lo que se denomina como *transfer learning*). Realmente, aunque el concepto de *embeddings* aparece en 2003 [8] no se popularizaron hasta que en 2013 varios investigadores de Google desarrollan Word2Vec [9], un software que ofrece un algoritmo para obtener *embeddings* y, a parte, un vocabulario de palabras extenso representadas en *embeddings* ya entrenados para tareas generales que se pueden reentrenar para adaptarlos a una tarea concreta. Desde entonces se han desarrollado una gran cantidad de métodos diferentes para generar *word embeddings* y diferentes modelos disponibles de *embeddings* generados, entre los que destacan:

Word2Vec: Es un método probabilístico de ventanas que incluye dos variantes: CBOW y Skip-Gram. En la primera se intenta predecir una palabra dado el contexto de esta. Skip-Gram, al contrario, dada una palabra intenta predecir la distribución (de probabilidad) de palabras de contexto.

GloVe: es un modelo de mínimos cuadrados ponderados que se entrena con recuentos globales de coocurrencia palabra–palabra. Estos recuentos se calculan en una matriz de coocurrencia, lo que puede ser costoso, pero solo es un cálculo inicial. Aprovecha de manera eficiente la información estadística global entrenando solo en los elementos distintos de cero de la matriz y produce un espacio vectorial con una subestructura

significativa. Se ha probado que es mejor que Word2Vec para la analogía de palabras y más rápido en un mismo corpus de texto [10].

A parte de los dos anteriores existen otros métodos como FastText [63], LexVec [64], Context2Vec [65], etc. Sin embargo, Word2Vec posiblemente sea el más conocido por lo que se profundizará en su implementación. Como se ha comentado anteriormente, existen dos variantes principales del método de aprendizaje Word2Vec (CBOW y Skip-Gram [44]):

CBOW

El modelo de bolsa continua de palabras o *Continuous Bag of Words* (CBOW) se basa en predecir una palabra a partir de su contexto. Se parte de una palabra o de un conjunto de ellas codificadas en representación *One-Hot Encoding* (se usa la notación $\mathbf{x}^{(c)} \in \mathbb{R}^{|\mathcal{V}|}$ para denotar la palabra central a predecir).

La incógnitas serán dos matrices, $V \in \mathbb{R}^{m \times |\mathcal{V}|}$ y $U \in \mathbb{R}^{|\mathcal{V}| \times m}$, donde: m es un tamaño arbitrario que define la dimensión del vector *embedding*; \mathcal{V} el vocabulario del corpus de textos; V la matriz de palabras de entrada, de modo que la i -ésima columna de V es el vector de tamaño m incorporado para la palabra w_i cuando es una entrada para este modelo (se denota a este vector como \mathbf{v}_i); y, de forma similar, U es la matriz de palabras de salida, de modo que j -ésima fila de U es un vector *embedding* de tamaño m para la palabra w_j cuando es una salida del modelo (se denota a esta fila de U como \mathbf{u}_j).

Aplicando dicha notación, se usan las palabras de contexto de una ventana de tamaño $2d$ con respecto a la palabra central (es decir, $\{\mathbf{x}^{(c-d)}, \dots, \mathbf{x}^{(c-1)}, \mathbf{x}^{(c+1)}, \dots, \mathbf{x}^{(c+d)}\} \subset \mathbb{R}^{|\mathcal{V}|}$) para generar los vectores *embeddings* de contexto $\{\mathbf{v}_{c-d} = V\mathbf{x}^{(c-d)}, \dots, \mathbf{v}_{c-1} = V\mathbf{x}^{(c-1)}, \mathbf{v}_{c+1} = V\mathbf{x}^{(c+1)}, \dots, \mathbf{v}_{c+d} = V\mathbf{x}^{(c+d)}\} \subset \mathbb{R}^m$. Una vez obtenidos, se calcula un vector promedio $\hat{\mathbf{u}} = \frac{\mathbf{v}_{c-d} + \dots + \mathbf{v}_{c-1} + \mathbf{v}_{c+1} + \dots + \mathbf{v}_{c+d}}{2d}$ para generar otro vector $\mathbf{z} = U\hat{\mathbf{u}} \in \mathbb{R}^{|\mathcal{V}|}$. A continuación se aplica la función *softmax* al vector \mathbf{z} para obtener el vector de probabilidades $\hat{\mathbf{y}} \in \mathbb{R}^{|\mathcal{V}|}$. Se ilustra dicho procedimiento en la figura 3.6a.

Para evaluar el error del proceso y entrenar el modelo (cuyos parámetros, como se ha comentado, son las matrices V y U) se utiliza la entropía cruzada (sección 4.2.3), entre los vectores $\hat{\mathbf{y}}$ y \mathbf{y} , donde este último es el verdadero vector *One-Hot Encoding* que representa a la palabra central.

Así, la representación en *embeddings* que se estaba buscando es la generada al aplicar la matriz V al vector de la palabra de entrada *One-Hot Encoding*.

Skip-Gram

El modelo de *Skip-Gram* se basa, al contrario que CBOW, en predecir las palabras de contexto de una palabra central dada. Manteniendo la notación vista anteriormente y de forma análoga que en CBOW, las incógnitas serán las dos matrices, $V \in \mathbb{R}^{m \times |\mathcal{V}|}$ y

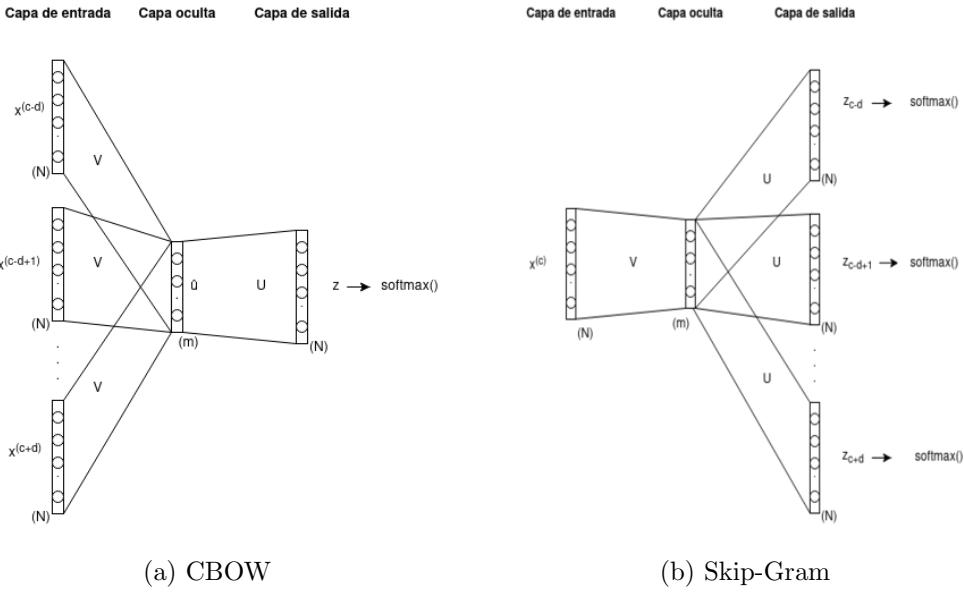


Imagen 3.6: Esquema de funcionamiento de las variantes de Word2Vec con la notación vista en la descripción anterior. Imagen adaptada de [48].

$$U \in \mathbb{R}^{|\mathcal{V}| \times m}.$$

Para ajustar el modelo se parte de una palabra codificada en representación *One-Hot Encoding* denotada por $\mathbf{x}^{(c)} \in \mathbb{R}^{|\mathcal{V}|}$. A partir de ella, se obtiene el vector *embedding* de dicha palabra ($\mathbf{v}_c = V\mathbf{x}^{(c)} \in \mathbb{R}^m$). Con él, se genera el vector de puntuación $\mathbf{z} = U\mathbf{v}_c \in \mathbb{R}^{|\mathcal{V}|}$, y, de forma análoga a CBOW, se le aplica la función *softmax* para obtener, en este caso, un conjunto de vectores de probabilidades de las palabras de contexto $\{\hat{\mathbf{y}}^{(c-d)}, \dots, \hat{\mathbf{y}}^{(c-1)}, \hat{\mathbf{y}}^{(c+1)}, \dots, \hat{\mathbf{y}}^{(c+d)}\}$. Se ilustra, también, dicho procedimiento en la figura 3.6b.

Para ajustar el modelo, esta vez se supone que todas las palabras de contexto son independientes entre ellas, por lo que se puede aplicar la entropía cruzada de nuevo entre cada $\hat{\mathbf{y}}^{(c-i)}$ y $\mathbf{y}^{(c-i)}$ donde $i \in \{d, d-1, \dots, 1, -1, \dots, -(d-1), -d\}$ e $\mathbf{y}^{(c-i)}$ es la palabra de contexto real en la posición que indique $c - i$ en la ventana. La función de pérdida será la suma de cada uno de estos valores de entropía.

De nuevo, la representación en *embeddings* que interesa es la generada al aplicar V al vector *One-Hot Encoding* de la palabra de entrada.

3.5.2. Redes neuronales recurrentes (RNNs)

Las redes neuronales recurrentes son una familia de redes neuronales que permiten procesar datos de naturaleza secuencial, es decir, datos en los cuales el orden, en el que se disponen, es información de valor. Además, permiten una entrada de longitud varia-

ble. Por ejemplo, su uso está bastante extendido para clasificar videos, predecir el tiempo meteorológico, predicciones de acciones en bolsa o en el Procesamiento del Lenguaje Natural.

La estructura básica de una red neuronal recurrente es la ilustrada en la imagen 3.7. Como se puede apreciar, la red está formada por capas ocultas, donde la capa $\mathbf{h}^{(i)}$ está asociada al instante de tiempo i . Cada una de estas capas ocultas contiene un número determinado de neuronas, cada una de las cuales realiza una operación matricial lineal en sus entradas seguida de una operación no lineal. Además, como se ilustra, en cada instante de tiempo i , se le proporcionan dos entradas a la capa oculta $\mathbf{h}^{(i)}$: la salida de la capa oculta anterior, $\mathbf{h}^{(i-1)}$, (asociada al instante de tiempo anterior, $i - 1$) y la entrada (información del exterior) de la secuencia que se corresponde a ese instante de tiempo, $\mathbf{x}^{(i)}$. En el rectángulo morado se indica el interior de la red neuronal recurrente asociada al instante de tiempo 2.

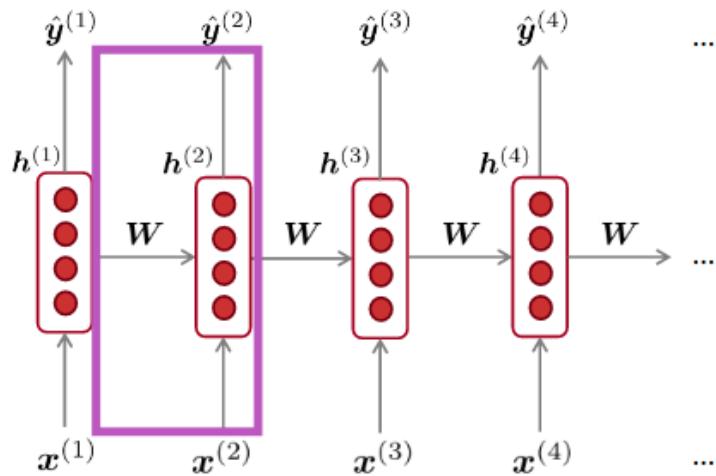


Imagen 3.7: Esquema de una red neuronal recurrente donde se muestran cuatro instantes en el tiempo. [44]

La entrada proporcionada por la salida de la capa oculta anterior se multiplica por una matriz de pesos $W_h \in \mathbb{R}^{n \times n}$ (si se considera que el tamaño de la capa oculta es de n neuronas) y la entrada que se proporciona del exterior, por una matriz de pesos $W_x \in \mathbb{R}^{n \times d}$ (si se considera que el tamaño de la entrada es de dimensión d , es decir, $\mathbf{x}^{(i)} \in \mathbb{R}^d$). Los resultados de salida de la red en cada instante de tiempo i , $\hat{\mathbf{y}}^{(i)} \in \mathbb{R}^m$, son opcionales, y, según el propósito, serán necesarios devolverlos o no para un determinado instante de tiempo. En el caso de que sea necesario devolver una salida en el instante de tiempo i , la salida producida por la capa oculta $\mathbf{h}^{(i)}$ se multiplica por una matriz de pesos $U \in \mathbb{R}^{m \times n}$ y el resultado se le proporciona a una función de activación para obtener el resultado de salida.

Se considera el siguiente problema: dada una secuencia de palabras de entrada de tamaño arbitrario $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)}\}$ (se supone tamaño m), donde cada palabra pertenece al vocabulario \mathcal{V} , se pretende construir un modelo que genere la siguiente palabra de dicha secuencia, es decir, la palabra $\mathbf{x}^{(m+1)}$. Las palabras pueden estar representadas por alguna técnica de *One-Hot Encoding* y la primera tarea del modelo será obtener una representación en *embeddings* de dicha entrada; o se puede partir de una representación en *embeddings* del vocabulario entrenado previamente, como, por ejemplo, los proporcionados por Word2Vec, explicados en la sección 3.5.1). Una vez lograda esta representación, el siguiente paso será introducir el primer *embedding* en la capa oculta $\mathbf{h}^{(1)}$, que recibe como entrada además la salida de una capa oculta inicial, $\mathbf{h}^{(0)}$, que suele ser un parámetro de la red o ser el vector $\mathbf{0}$. Así, la salida proporcionada por la capa oculta $\mathbf{h}^{(1)}$ viene dada por:

$$\mathbf{h}^{(1)} = \sigma(W_h \mathbf{h}^{(0)} + W_x \mathbf{x}^{(1)} + \mathbf{b}_1)$$

donde \mathbf{b}_1 es el término de sesgo y la función σ es una función de activación.

Siguiendo esta misma lógica, el siguiente paso será introducir el siguiente *embedding* ($\mathbf{x}^{(2)}$) en la siguiente capa oculta $\mathbf{h}^{(2)}$. Así la salida de esta capa oculta viene dada por:

$$\mathbf{h}^{(2)} = \sigma(W_h \mathbf{h}^{(1)} + W_x \mathbf{x}^{(2)} + \mathbf{b}_1)$$

Generalizando este procedimiento a todas las capas ocultas de la red se tiene que, para un instante de tiempo t :

$$\mathbf{h}^{(t)} = \sigma(W_h \mathbf{h}^{(t-1)} + W_x \mathbf{x}^{(t)} + \mathbf{b}_1)$$

En el instante de tiempo m puede ser necesario devolver una salida del modelo. Así, la salida producida por la capa oculta $\mathbf{h}^{(m)}$ se multiplica por una matriz de pesos U , se le añade un término de sesgo \mathbf{b}_2 y el resultado se le proporciona a una función *softmax* sobre todo el vocabulario para obtener una predicción de la palabra de salida $\hat{\mathbf{y}} = \mathbf{x}^{(m+1)}$.

$$\hat{\mathbf{y}}^{(m)} = \text{softmax}\left(U \mathbf{h}^{(m)} + \mathbf{b}_2\right)$$

Se ilustra dicho procedimiento con la imagen 3.8.

Así se ha visualizado el funcionamiento de las redes neuronales recurrentes para un ejemplo sencillo de modelado del lenguaje.

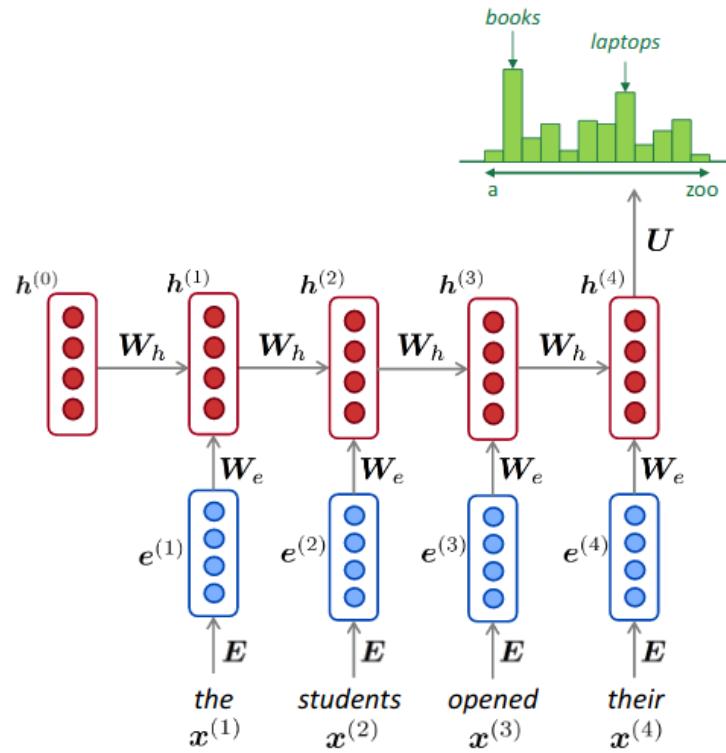


Imagen 3.8: Modelo generativo de texto mediante una red neuronal recurrente en el instante $m = 4$. [44]

El uso de las redes neuronales recurrentes para el problema de generación de texto viene motivado por los siguientes aspectos [44]:

- Las redes neuronales recurrentes pueden procesar cualquier secuencia de entrada de tamaño arbitrario.
- Las redes neuronales recurrentes, en teoría, para un instante t de tiempo hacen uso de información procesada de muchos instantes de tiempo anteriores, lo que permite a la red entrenar con información de contexto de las palabras u oraciones anteriores.
- Además, el tamaño del modelo generado por las redes recurrentes no aumenta para una entrada más larga, es decir, las matrices de parámetros del modelo, W_h, W_x y U , y los vectores de parámetros, \mathbf{b}_1 y \mathbf{b}_2 , tienen un tamaño fijo durante todo el entrenamiento del modelo.
- Los parámetros del modelo, pesos y sesgo, son invariantes durante todos los instantes de tiempo, por lo que hay simetría en cómo se procesa la entrada. Así los parámetros solo serán modificados al obtener la señal de error y realizar la retropropagación de dicho error y la consecuente optimización de parámetros.

Sin embargo, los modelos basados en redes neuronales recurrentes poseen varias des-

ventajas:

- El entrenamiento de una red recurrente es bastante más lento que el de una red neuronal artificial básica ya que, al necesitar los resultados de una capa oculta anterior como entrada de la capa oculta posterior, no se puede paralelizar el proceso.
- En la práctica resulta complicado, en un instante t acceder a la información procesada de muchos instantes de tiempo anteriores.

En entrenamiento de dichas redes serán objeto de estudio del capítulo 4, concretamente de la sección 4.5.

3.5.3. Redes neuronales recurrentes con celdas (RNNs)

Long Short-Term Memory (LSTM)

Como se comentó en la lista de desventajas de las redes neuronales recurrentes, es complicado en la práctica que, en un instante t , se pueda acceder a la información procesada algunos instantes de tiempo anteriores. Esto es debido a que en la fase de entrenamiento, la contribución de los valores de gradiente se desvanece gradualmente a medida que se propagan a los instantes de tiempo anteriores. Para solucionar éste problema surge un tipo de redes neuronales recurrentes, denominadas *Long Short-Term Memory* (LSTM) [17].

Este tipo de arquitectura incorpora a cada estado oculto $\mathbf{h}^{(t)}$ una celda de estado de la misma longitud, que se denota por $\mathbf{c}^{(t)}$ y cuya función es almacenar información de largo plazo. La red decide borrar, escribir o leer de dichas celdas y estas acciones están controladas por puertas o *gates* de igual longitud que las celdas. A diferencia de los parámetros de una red neuronal recurrente básica, los vectores *gates* son dinámicos, es decir, varían según el instante t de la red y dicho valor se calcula según el contexto. Formalmente: dada una secuencia de palabras de entrada de tamaño arbitrario $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)}\}$ (se supone tamaño m), donde cada palabra pertenece al vocabulario \mathcal{V} . Así se dispone de una secuencia de estados ocultos $\{\mathbf{h}^{(0)}, \mathbf{h}^{(1)}, \dots, \mathbf{h}^{(m)}\}$ y una secuencia de celdas de estado $\{\mathbf{c}^{(0)}, \mathbf{c}^{(1)}, \dots, \mathbf{c}^{(m)}\}$. Además se dispone de tres puertas: la puerta de olvido (*forget gate*), $f^{(t)}$, que controla que se mantiene o se olvida con respecto a la celda de estado anterior; la puerta de entrada (*input gate*), $i^{(t)}$, que controla que partes de la información se incluyen en la nueva celda de estado; y la puerta de salida (*output gate*), $o^{(t)}$, que controla que información de la celda se incorpora al estado oculto. Se incluye a continuación las funciones que las definen, donde σ es la función sigmoide:

$$\begin{aligned} f^{(t)} &= \sigma(W_f h^{(t-1)} + U_f x^{(t)} + b_f) \\ i^{(t)} &= \sigma(W_i h^{(t-1)} + U_i x^{(t)} + b_i) \\ o^{(t)} &= \sigma(W_o h^{(t-1)} + U_o x^{(t)} + b_o) \end{aligned}$$

Además se dispone de las siguientes funciones que las relacionan:

$$\hat{c}^{(t)} = \tanh(W_c h^{(t-1)} + U_c x^{(t)} + b_c)$$

que incluye el contenido de la nueva celda.

$$c^{(t)} = f^{(t)} \otimes c^{(t-1)} + i^{(t)} \otimes \hat{c}^{(t)}$$

que olvida el contenido de la última celda que corresponda y que escribe el nuevo contenido de entrada. La operación \otimes indica el producto elemento a elemento.

$$h^{(t)} = o^{(t)} \otimes \tanh c^{(t)}$$

que proporciona información de la celda al estado oculto.

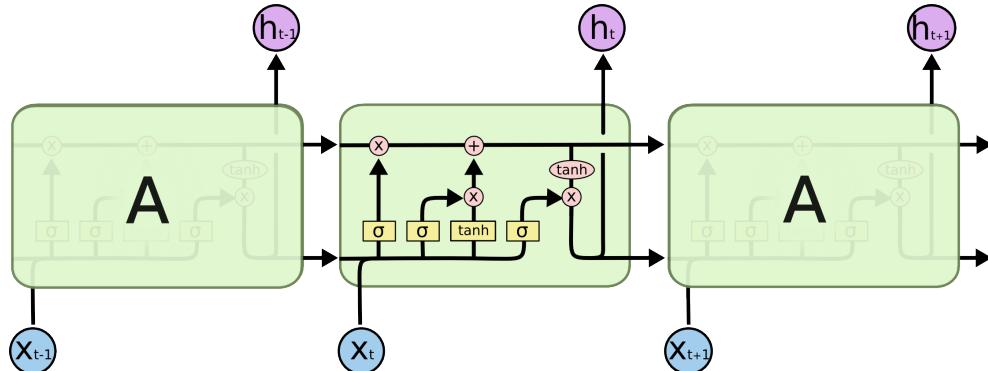


Imagen 3.9: Ilustración que muestra un esquema de la arquitectura de una LSTM. [46]

Con este tipo de arquitectura de red neuronal recurrente se consigue que se mantenga información procesada a largo plazo de la secuencia, lo que provee a la red de mayor información de contexto.

Gated Recurrent Units (GRU)

Esta arquitectura de red neuronal recurrente, propuesta en 2014 [18], simplifica la arquitectura de las LSTM, eliminando las celdas de estado $c^{(t)}$ que aparecían en las redes LSTM. La red, aun así, usa las puertas o *gates* para controlar el flujo de información. Formalmente: dada una secuencia de palabras de entrada (de tamaño arbitrario m)

$\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$, donde cada palabra pertenece al vocabulario \mathcal{V} , se dispone de una secuencia de estados ocultos $\{h^{(0)}, h^{(1)}, \dots, h^{(m)}\}$. Además, esta vez, se dispone de dos puertas: la puerta de actualización, $z^{(t)}$, (*update gate*), que controla qué información del estado oculto se mantiene o se actualiza; y la puerta de reinicio, $r^{(t)}$, (*reset gate*), que controla qué partes de la información del anterior estado se usa para obtener la nueva información. Se incluye a continuación las funciones que las definen, donde σ es la función sigmoide:

$$\begin{aligned} z^{(t)} &= \sigma(W_z h^{(t-1)} + U_z x^{(t)} + b_z) \\ r^{(t)} &= \sigma(W_r h^{(t-1)} + U_r x^{(t)} + b_r) \end{aligned}$$

Además se dispone de las siguientes funciones que las relacionan:

$$\hat{h}^{(t)} = \tanh(W_h(r^{(t)} \otimes h^{(t-1)}) + U_h x^{(t)} + b_h)$$

que selecciona qué partes del estado oculto anterior son útiles y lo usa para obtener el nuevo estado oculto.

$$h^{(t)} = (1 - z^{(t)}) \otimes h^{(t-1)} + z^{(t)} \otimes \hat{h}^{(t)}$$

que indica que el nuevo estado oculto es una combinación entre la información calculada por $\hat{h}^{(t)}$ y el anterior estado oculto.

Las ventajas de dicha arquitectura frente a las LSTM es que son más simples y por tanto más rápidas de entrenar, al poseer menos parámetros que ajustar. Es habitual, si dado un problema, existen multitud de dependencias contextuales, usar redes LSTM ya que suelen mantener información a largo plazo de manera más habitual que las redes GRU, pero esta ventaja provoca un coste computacional más elevado.

Existen, además de las expuestas, una gran variedad de arquitecturas de redes neuronales recurrentes, como las redes neuronales recurrentes bidireccionales o las redes neuronales recurrentes multicapa. Sin embargo, desde su introducción en 2017, una nueva arquitectura de red neuronal se ha convertido en la herramienta más utilizada en el campo del *Natural Language Processing*: los *transformers*.

3.5.4. Transformers

Motivación y conceptos previos

En 2014, el artículo *Sequence to Sequence Learning with Neural Networks* [19] introdujo *Sequence-to-sequence* o *Seq2Seq*, que es un modelo formado por dos redes neuronales

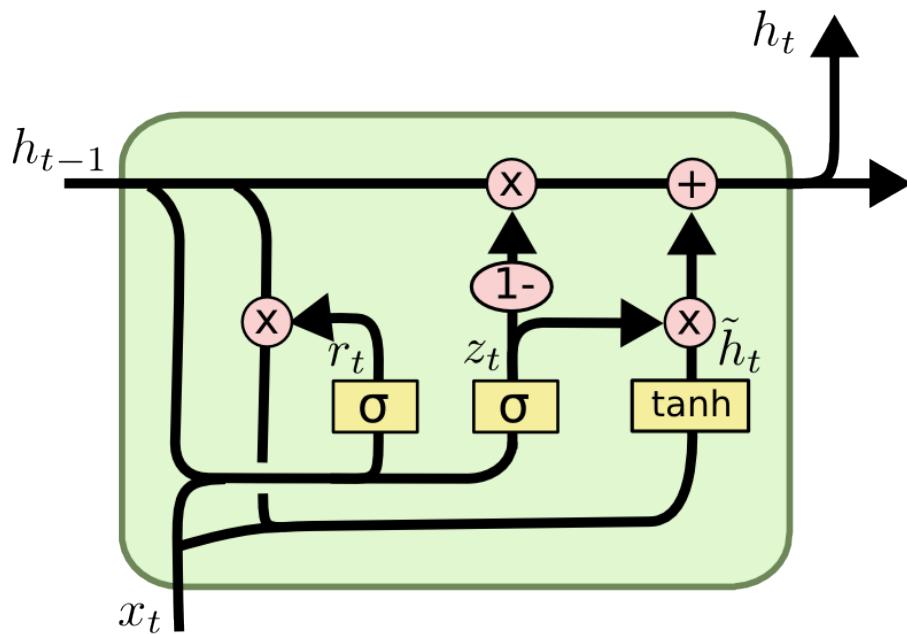


Imagen 3.10: Ilustración que muestra un esquema de la arquitectura de una GRU. [46]

recurrentes: un *encoder* (codificador), que toma la secuencia de entrada del modelo y la codifica en un vector de contexto de tamaño fijo; y un *decoder* (decodificador), que usa el vector de contexto como una “semilla” a partir de la cual generar una secuencia de salida.

Sin embargo, dada una secuencia de entrada al modelo *Seq2Seq*, el *encoder* transforma toda la información de contexto en un solo vector de salida que, como se ha visto en la sección 3.5.2, puede resultar que mantenga poca información de las primeras palabras dicho contexto. Esto motiva la introducción de una técnica que solucione este problema: los mecanismos de atención. Dichos mecanismos permiten al *decoder* decidir qué palabras de entrada son importantes en cualquier momento del proceso.

Se puede describir un mecanismo de atención, en una arquitectura *encoder-decoder*, de la siguiente forma. Se supone una secuencia de palabras (de tamaño arbitrario m) $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$, donde cada palabra pertenece al vocabulario \mathcal{V} , sabiendo que se dispone de un *encoder* donde, para cada instante de tiempo t , se tiene un estado oculto $\mathbf{h}^{(t)} \in \mathbb{R}^d$ y, además, para cada instante de tiempo t , se tiene otro estado oculto, $\mathbf{s}^{(t)} \in \mathbb{R}^d$, pero en el *decoder*. Entonces, para cada instante de tiempo t se obtiene un vector de atención de la entrada, $\mathbf{e}^{(t)} = [\mathbf{s}^{(t)\top} \mathbf{h}^{(1)}, \mathbf{s}^{(t)\top} \mathbf{h}^{(2)}, \dots, \mathbf{s}^{(t)\top} \mathbf{h}^{(m)}] \in \mathbb{R}^m$. A continuación, se aplica la función *softmax* sobre dicho vector obteniendo una distribución de probabilidad $\boldsymbol{\alpha}^{(t)} \in \mathbb{R}^m$. Y, por último, se utiliza $\boldsymbol{\alpha}^{(t)}$ para obtener una suma ponderada de los estados ocultos del *encoder*, $\boldsymbol{\alpha}_t = \sum_{i=1}^m \alpha_i^{(t)} \mathbf{h}^{(i)}$.

Otro problema contra el que se luchaba hasta la fecha era que el entrenamiento de las redes neuronales recurrentes, al ser secuencial, era bastante lento.

Con estas motivaciones surge la idea de *transformer*. Un *transformer* [21] se basa en la conexión de un codificador y un decodificador mediante un mecanismo de atención, prescindiendo de recurrencias y otros artificios, dando lugar a un modelo más paralelizable y con un tiempo de entrenamiento menor.

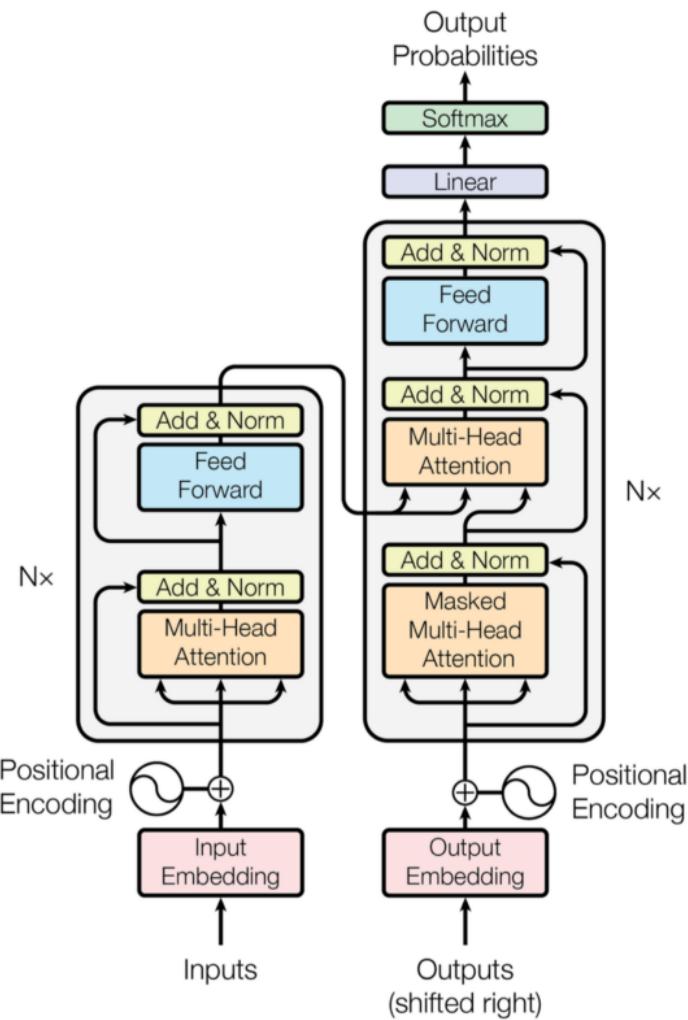


Imagen 3.11: Ilustración que muestra un esquema de la arquitectura de un *transformer*, mostrando solo una sola capa *encoder* y otra *decoder* [21].

Arquitectura de un *transformer*

Entrando en los detalles de su arquitectura, un *transformer* sigue la base formada por un codificador y decodificador (*encoder-decoder*) cuyas capas están totalmente conectadas y usan un mecanismo de auto-atención (*self-attention*).

El codificador (*encoder*) se compone de una pila de $N = 6$ capas idénticas (tal y como se propone en el artículo original [21]). Cada una de dichas capas tiene dos subcapas: la primera, un mecanismo de auto-atención de múltiples cabezales (*multi-head self-attention*); y la segunda, una red neuronal *feed-forward* completamente conectada en función de la posición. Se emplea además una conexión residual alrededor de cada una de las dos subcapas, seguida de una normalización. Para hacer más fácil el trabajo, todas las subcapas del modelo, así como las capas de embeddings, producen salidas de dimensión $d_{model} = 512$.

El decodificador (*decoder*) también está formado por una pila de $N = 6$ capas idénticas. La diferencia con el codificador es que el decodificador inserta una tercera subcapa, un mecanismo de atención de múltiples cabezales *multi-head attention* sobre la salida del codificador. Pero también, de forma similar al codificador, se emplean conexiones residuales alrededor de cada una de las subcapas, seguidas de una normalización. Se enmascara la subcapa (*multi-head self-attention*) del decodificador para evitar que se puedan mostrar posiciones posteriores, con lo que se asegura que las predicciones para la posición i puedan depender solo de las salidas conocidas en posiciones menores que i .

Esta es la estructura general de un *transformer*, pero ¿qué es el mecanismo de auto-atención de múltiples cabezales o *multi-head self-attention*?.

Tal y como se explica en el artículo original [21], una función de atención consiste en mapear una consulta (*query*), que es la entrada de esta función de atención, a un conjunto de pares clave-valor (*key-value*).

El *transformer* utiliza la función de atención denominada *Multi-Head Self-Attention*. La entrada consta de consultas y claves (vectores de dimensión d_k) y valores (vectores de dimensión d_v). Estos vectores de entrada se han creado multiplicando los *embeddings* de entrada al *transformers* por tres matrices que se ajustan durante el proceso de entrenamiento.

A continuación se calcula un vector de similitud entre el vector *query* y cada uno de los vectores *key*. Para ello se hace uso del producto escalar. De esta forma, se calculan los productos escalares del vector *query* con todos los vectores *keys*, se dividen por $\sqrt{d_k}$ y se aplica una función *softmax* para obtener una distribución de probabilidad que indica el peso de cada uno de los valores *values*.

Una vez conseguido esto, se multiplica cada vector *value* por la puntuación correspondiente de la distribución de probabilidad anterior. Por último, se suman todos los vectores *values* obteniendo la salida deseada de atención.

En la práctica, se calcula la función de atención en un conjunto de consultas simultáneamente en un matriz Q . Las claves y los valores también se calculan en las matrices K y V respectivamente. La matriz de salidas es el resultado de:

$$\text{Atencion}(Q, K, V) = \text{softmax} \left(\frac{QK^\top}{\sqrt{d_k}} \right) V$$

Hasta aquí se ha construido solo una capa *self-attention*. Sin embargo, la arquitectura real usa varias capas o cabezales (lo que da nombre al bloque, *multi-head self-attention*). Esto permite ampliar la capacidad del modelo para centrarse en diferentes posiciones y no solo en una.

De esta manera y usando dicha definición, se define la atención *multi-head self-attention* como:

$$\text{MultiHead}(Q, K, V) = \text{Concatenacion}(\text{head}_1, \dots, \text{head}_h)W^O$$

donde $\text{head}_i = \text{Atencion}(QW_i^Q, KW_i^K, VW_i^V)$. Es decir, en lugar de realizar una única función de atención con claves, valores y consultas, se proyectan las consultas, claves y valores h veces con diferentes matrices que se ajustan durante el entrenamiento, a dimensiones d_k , d_k y d_v , respectivamente. En cada una de estas versiones proyectadas de consultas, claves y valores, se aplica la función de atención en paralelo, produciendo h vectores de salida d_v -dimensionales. Sin embargo, solo se debe devolver una salida que servirá como entrada del bloque neuronal *feed-forward* siguiente. Para resolverlo se concantenan los distintos vectores formando una matriz y se aplica otra matriz de proyección W^O dando como resultado el vector de atención final. Las matrices usadas son: $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ con $i \in \{1, \dots, h\}$ y $W^O \in \mathbb{R}^{hd_v \times d_{model}}$. Se muestra un esquema intuitivo en la figura 3.12.

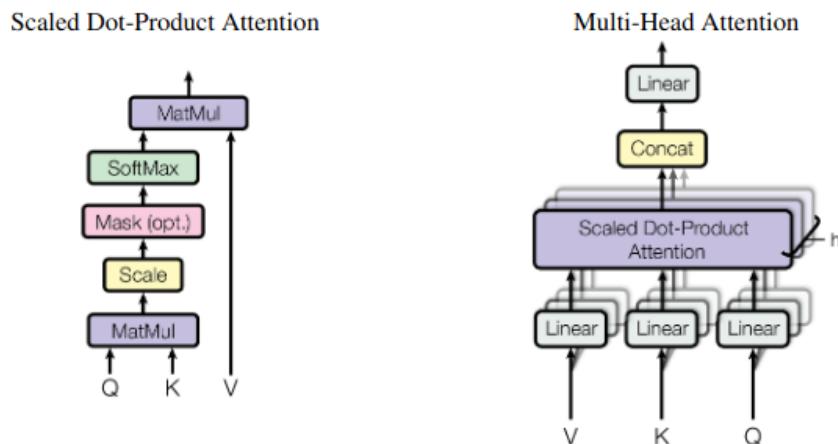


Imagen 3.12: Ilustración que muestra un esquema de la función de *multi-head self-attention* de un *transformer*.

En definitiva, el bloque de *Multi-Headed Attention* genera un vector de atención para

cada palabra de la secuencia de entrada, que obtiene las relaciones de contexto entre estas palabras.

3.5.5. Ejemplos de modelos de lenguaje actuales

La arquitectura mostrada en la sección anterior ha sustentado las bases de los modelos generativos del lenguaje más actuales, dejando a un lado las redes neuronales recurrentes. Esta sección se centra en estudiar los modelos generativos del lenguaje que han sido más populares en los últimos años.

Primero se introducirán los modelos que introdujeron las representaciones de palabras profundamente contextualizadas, como TagLM y ELMo. Seguidamente se presentarán los distintos modelos generativos que han tenido más éxito en éstos años: GPT, BERT y GPT-2, haciendo mención del último gran avance del campo de los modelos generativos del lenguaje, como es GPT-3.

TagLM y ELMo

La representación de palabras vista hasta el momento (sección 3.5.1) está limitada por un problema fundamental: la polisemia de las palabras, es decir, existen palabras con diversos significados. Una solución a este problema podría ser contar con tantas representaciones de la misma palabra según el significado que se tome. Sin embargo, lo que de verdad interesa es captar el significado de la palabra según el contexto en que se encuentre en el caso particular donde se trabaje.

Una primera aproximación a esta idea intuitiva fue la proporcionada por el modelo TagLM [20], que es entrenado con un enfoque semi-supervisado, es decir: dado un gran número de datos sin etiquetar, se entrena un modelo de *embedding* tradicional, como Word2Vec, que es independiente del contexto, y, al mismo tiempo se entrena un modelo del lenguaje recurrente que sí posee características de contexto. Ambos modelos devuelven unos vectores que forman la representación final de las palabras incluyendo así características de contexto. El esquema del modelo TagLM se encuentra en la figura 3.13.

Sin embargo, el verdadero modelo revolucionario, que introdujo la representación de palabras profundamente contextualizadas, fue el modelo ELMo [22], introducido un año después como mejora del modelo TagLM. Este modelo permite tener en cuenta características como la sintaxis de la palabra y cómo se utiliza en diferentes contextos. Hace uso de LSTM bidireccionales (tiene en cuenta las palabras que aparecen antes y después de una palabra dada).

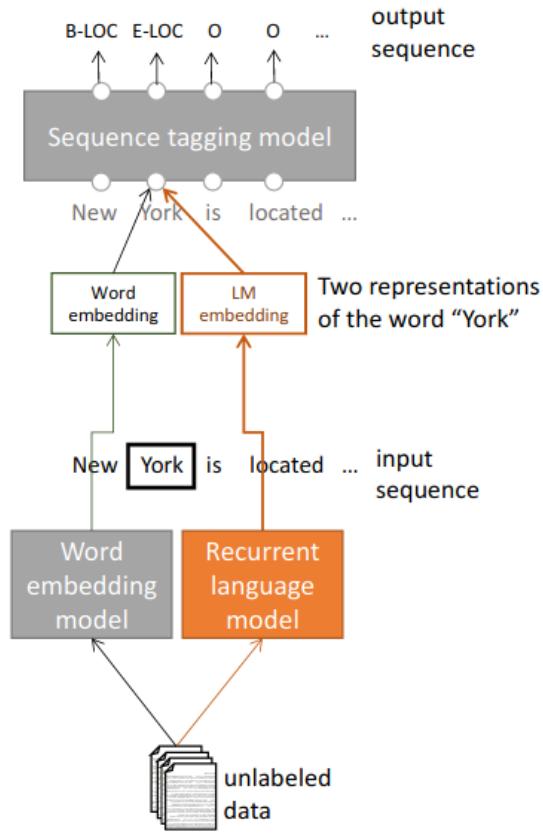


Imagen 3.13: Ilustración que muestra los componentes principales de TagLM. El componente del modelo de lenguaje (en naranja) se utiliza para aumentar la representación del token de entrada en un modelo de etiquetado de secuencia tradicional (en gris) [20].

GPT

En mayo de 2018 llega a la luz el modelo *Universal Language Model Fine-tunning for Text Classification* (ULMFiT) [23], un método de aprendizaje por transferencia (*transfer learning*) que, a pesar de inspirarse en la clasificación de textos, puede ser aplicado a cualquier tarea de NLP, lo que introduce técnicas clave para ajustar un modelo de lenguaje. Meses después sale a la luz el modelo de OpenAI, GPT [24]. Dicho modelo se basa en un *transformer* básico (unidireccional), pre-entrenado usando modelado de lenguaje en un gran corpus con dependencias de largo alcance, el *Toronto Book Corpus*.

El modelo GPT parte de un gran corpus de texto no supervisado: $\mathcal{U} = \{u_1, \dots, u_n\}$. El objetivo es maximizar la probabilidad $L_1(\mathcal{U}) = \sum_i \log P(u_i | u_{i-k}, \dots, u_{i-1}; \Theta)$, donde k es el tamaño de la ventana de contexto, y la probabilidad condicional P se modela usando una red neuronal con parámetros Θ .

Se utilizó un decodificador de transformador multicapa *multi-layer Transformer decoder* para el modelo de lenguaje (concretamente de 12 capas con 768 estados) teniendo que

ajustar cerca de 117 millones de parámetros. Este modelo aplica una operación de autoatención de múltiples cabezales (*multi-headed self-attention*) sobre los tokens de contexto de entrada seguida de redes neuronales *feed-forward* para producir una distribución de salida sobre los tokens de destino. Para su optimización se utilizó el optimizador Adam (descrito en la sección 4.3.9). Formalmente:

$$\begin{aligned} h_0 &= UW_e + W_p \\ h_l &= \text{transformer_block}(h_{l-1}) \quad \forall l \in \{1, \dots, n\} \\ P(u) &= \text{softmax}(h_n W_e^\top) \end{aligned}$$

donde $U = \{u_{-k}, \dots, u_{-1}\}$ es el vector de contexto de las palabras, n el número de capas, W_e la matriz de *embeddings* y W_p es la matriz de posición de los *embeddings*.

Después del preentrenamiento, se adaptan los parámetros a la tarea objetivo supervisada: dado conjunto de datos etiquetados \mathcal{C} , donde cada instancia consta de una secuencia de palabras de entrada, $\{x^{(1)}, \dots, x^{(m)}\}$, junto con una etiqueta y , se pasan a través del modelo previamente entrenado para obtener la activación del bloque del transformador final, h_l^m , que luego alimenta una capa de salida lineal adicional con los parámetros W_y para la predicción de y :

$$P(y|x^{(1)}, \dots, x^{(m)}) = \text{softmax}(h_l^m W_y)$$

Esto proporciona el siguiente objetivo a maximizar:

$$L_2(\mathcal{C}) = \sum_{(x,y)} \log P(u_i|x^{(1)}, \dots, x^{(m)})$$

Además, al incluir el modelado del lenguaje como un objetivo auxiliar del ajuste *fine-tuning* se produce una mejora en la generalización del modelo supervisado y se acelera la convergencia. Así pues, el objetivo es optimizar, para un cierto λ (en el artículo original $\lambda = 0,5$), lo siguiente:

$$L_3(\mathcal{C}) = L_2(\mathcal{C}) + \lambda * L_1(\mathcal{C})$$

A partir de aquí, dependiendo de la tarea a realizar, se necesitan algunas modificaciones del modelo o aplicarlo directamente a la tarea correspondiente. Dichas modificaciones se pueden apreciar en la imagen 3.14.

El código original se puede encontrar aquí: [GPT Code](#).

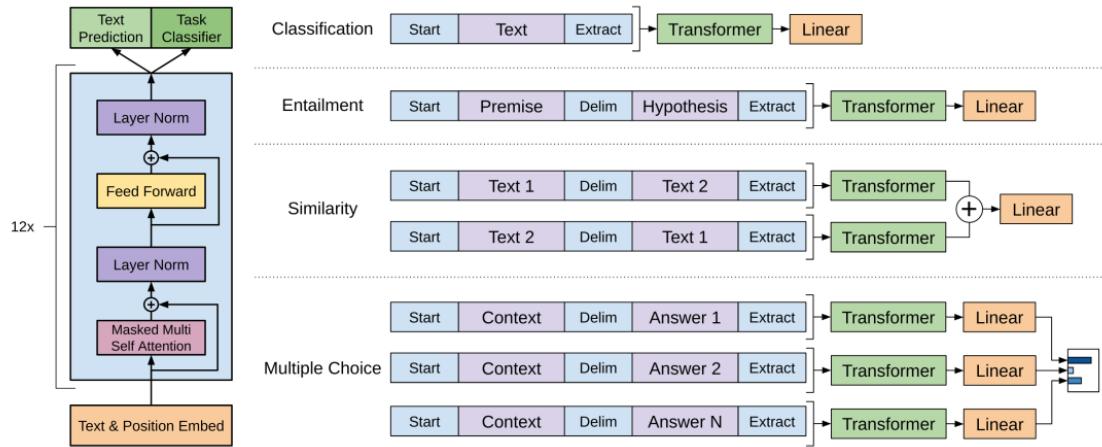


Imagen 3.14: Ilustración que muestra las transformaciones de la entrada del modelo GPT según la tarea a realizar [24].

BERT

En octubre de 2018 se presenta la primera versión del modelo BERT (*Bidirectional Encoder Representations from Transformers*) [25], que a diferencia de GPT (que se basaba en un decodificador *transformer* unidireccional), está diseñado para entrenar previamente representaciones bidireccionales profundas a partir de texto sin etiquetar y, posteriormente, reajustar los parámetros utilizando datos etiquetados para tareas más concretas.

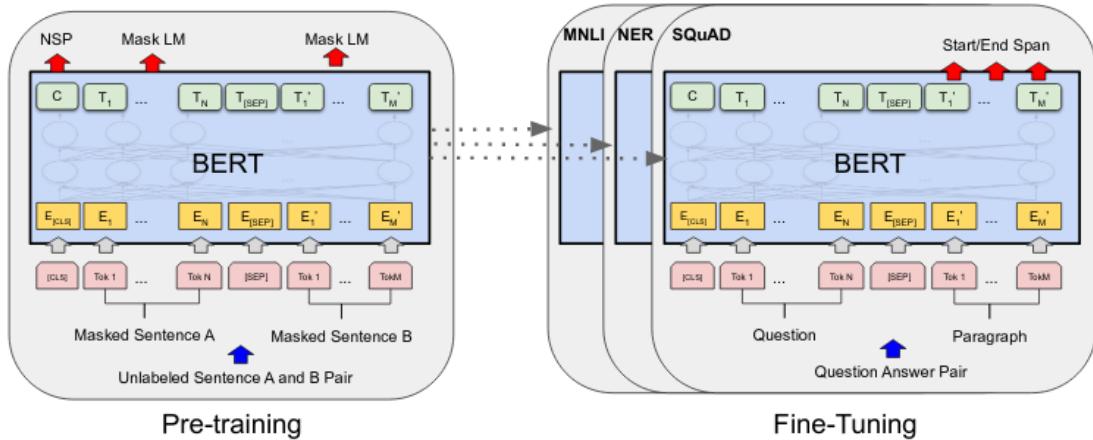


Imagen 3.15: Ilustración que muestra los procedimientos generales de preentrenamiento y ajuste para BERT [25].

Así la diferencia más significativa es la bidireccionalidad incluida en el modelo. Además, para el entrenamiento previo del modelo se realizan dos tareas no supervisadas: *maskedLM* (MLM), es decir, se enmascara un porcentaje de las palabras de entrada al azar y luego se

predicen dichas palabras enmascaradas; y *Next Sentence Prediction* (NSP), es decir, para aprender la relación entre frases se realiza en el entrenamiento una tarea de predicción binaria de la siguiente oración que puede generarse trivialmente. Concretamente, al elegir las oraciones *A* y *B* para cada ejemplo de entrenamiento previo, el 50 % de las veces *B* es la oración siguiente real que sigue a *A* (etiquetada como *IsNext*), y el 50 % de las veces es una oración aleatoria del corpus (etiquetada como *NotNext*).

El reajuste (*fine-tuning*) es bastante más simple y poco costoso computacionalmente, comparado con el preentrenamiento, gracias al mecanismo de auto-atención del *transformer*. Así, para la tarea que se corresponda, simplemente se ponen las entradas y salidas correspondientes y se ajustan los parámetros de principio a fin.

Se crearon dos modelos BERT: *BERT_{BASE}* y *BERT_{LARGE}*.

- *BERT_{BASE}*: Formado por 12 codificadores *transformer* de dimensión 768 y 12 cabezales de auto-atención (*self-attention heads*). Fue elegido para tener el mismo tamaño del modelo GPT para propósitos de comparación. En total es un modelo formado por cerca de 110 millones de parámetros.
- *BERT_{LARGE}*: Formado por 24 codificadores *transformer* de dimensión 1024 y 16 cabezales de auto-atención (*self-attention heads*). El modelo que se forma es de alrededor de 340 millones de parámetros. Las pruebas realizadas indican que mejora sustancialmente el modelo *BERT_{BASE}*.

A continuación muestra una imagen (3.16) que representa las diferencias significativas entre los modelos del lenguaje más destacados hasta entonces:

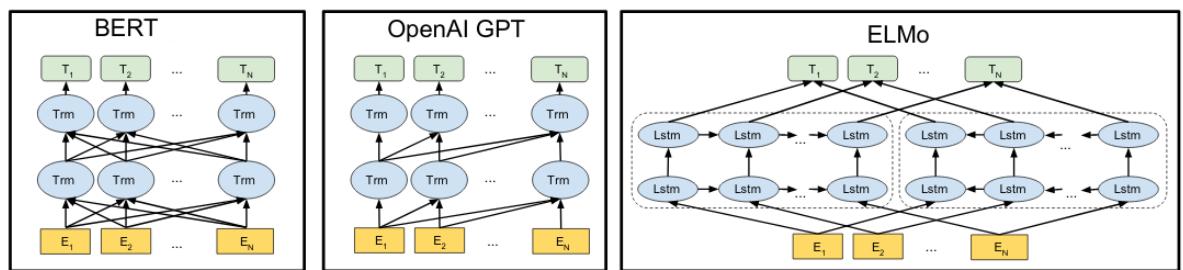


Imagen 3.16: A la izquierda: BERT, un modelo basado en los *transformers* bidireccionales. En el medio: GPT (3.5.5), basado en los *transformers* unidireccionales. A la derecha: ELMo (3.5.5), un modelo que usa concatenación de LSTMs bidireccionales [25].

GPT-2

Siguiendo la tendencia de entrenar modelos cada vez mayores, aparece en febrero de 2019 una versión del modelo GPT (sección 3.5.5) formada por muchos más parámetros

(cerca de 1542 millones de parámetros, es decir, más de 10 veces mayor que su predecesor) y entrenada con un conjunto de datos mucho más grande (40GB). Dicho modelo es llamado GPT-2 [26].

Realmente no hay cambios importantes en la arquitectura entre el modelo GPT y el modelo GPT-2, simplemente es más grande este último, concretamente está formado por 48 capas de decodificadores con 1600 estados cada uno frente a las 12 capas con 768 estados de su predecesor. Además se introdujo una subcapa de normalización y la subcapa de normalización que portaba GPT se desplaza hacia la entrada de cada bloque.

Una de las particularidades de GPT-2 es que ha demostrado ser capaz de realizar traducciones, resúmenes y hasta de contestar preguntas sobre el texto de forma bastante correcta sin necesidad de ser reajustado para ninguna de esas tareas, es decir, sin el proceso de *fine-tuning*, tan solo introduciendo como entrada información sobre que el modelo pueda comprender la tarea que tiene que realizar.

GPT-3

Más espectacular todavía es el modelo que salió a la luz en mayo de 2020, GPT-3 [27]. Dicho modelo está formado por 96 capas de dimensión 12288 dando lugar a más de 175000 millones de parámetros. Como la arquitectura es muy similar a la de GPT-2, su predecesor, no se entrará en más detalles.

Lo impresionante de la tendencia actual en el campo del *Natural Language Processing* es que se están desarrollando modelos solamente entrenados para generar texto, pero que son capaces de realizar tareas del lenguaje de manera increíblemente correcta, como resumir, traducir a múltiples idiomas, redactar artículos artificiales sobre cualquier tema que se le dé como entrada o incluso traducir texto natural a algún lenguaje de programación. A pesar de esto, existen todavía muchas limitaciones: una de ellas es que la comprensión que posee del mundo resulta no ser la adecuada en muchas situaciones y esto provoca que se pueda generar texto refiriéndose a circunstancias realmente improbables; también se ha convertido en un artefacto que permite generar cantidades desorbitadas de *fake news*, con los problemas que esto conlleva.

Variantes y otros modelos

El desarrollo de los modelos del lenguaje y su evolución en el crecimiento del número de parámetros entrenados no sería posible sin las investigaciones y mejoras que salen a la luz continuamente.

Un ejemplo claro son las diversas variantes de la arquitectura *transformer*, como, por ejemplo, *Transformer-XL* [28], una arquitectura que solventa los problemas (como la dependencia a largo plazo, la fragmentación del contexto o el uso de contexto de longitud

fija) que se encontraron en la arquitectura *transformer*. Meses después sale a la luz la arquitectura *Compressive Transformer* [29] que introduce una mejora al *Transformer-XL* compactando la información anterior y conservándola durante un mayor espacio de tiempo. A su vez, OpenAI desarrolla los *Sparse Transformer* [30] que, con la misma motivación, utiliza una mejora del mecanismo de atención para extraer patrones de secuencias mucho mayores. Dicha arquitectura es la que inspiró el modelo GPT-3 (3.5.5).

Por otro lado, es de obligatoria mención los modelos que, a pesar de no obtener una mayor fama, mejoraron a sus antecesores en estos años recientes:

- XLNet [31]: Salió a la luz en junio de 2019. Inspirado en los *Transformer-XL*, alcanzó el éxito al superar en 20 tareas a BERT.
- ERNIE [32]: Sale a la luz en marzo de 2019. Inspirado en la estrategia de enmascaramiento de BERT, aplica dicho enmascaramiento a frases y entidades para aprender tanto la información sintáctica como la semántica. Logra excelentes resultados en diversas tareas en chino, superando significativamente los métodos anteriores.
- ERNIE 2.0 [33]: Desarrollado en julio de 2019. Mejora de su predecesor, ERNIE, capacitado para aprender más tipos de tareas de forma continua en su entrenamiento. Supera a BERT y a XLNet en 16 tareas, incluidas tareas en chino como en inglés.
- T-NLG [34]: Publicado por Microsoft en febrero de 2020, fue el modelo más grande (17 mil millones de parámetros) hasta entonces entrenado, obteniendo mejoras en gran cantidad de tareas.

3.5.6. *Transfer Learning y Fine Tuning*

Una vez fijada la base, que ha permitido conducir el desarrollo teórico de los modelos generativos del lenguaje, culminando con algunos ejemplos de dichos modelos, se plantea ahora la cuestión de cómo utilizar dichos modelos en la práctica, para un problema concreto.

Una opción sería copiar la arquitectura desarrollada por los autores de los distintos modelos y entrenarlos desde cero, ajustando los diversos hiperparámetros, funciones de pérdida y optimizadores (véase las secciones 4.2 y 4.3 para más información) según las características del problema planteado. Sin embargo, se ha demostrado con anterioridad que los modelos, para que sean útiles y de buena calidad, están necesitados de un gran corpus de texto para su entrenamiento (en los mismos artículos de los autores de los modelos explicados se muestran resultados sobre esto: los datos y el tiempo necesarios para un correcto entrenamiento). Esto conlleva varios problemas: el tiempo de entrenamiento puede ser inmenso y se necesitan recursos computacionales que no están al alcance de la mayoría, así como un conjunto de datos suficientemente grande para ello, del que,

normalmente, no se dispone.

Esto se solventa usando el modelo previamente entrenado que es proporcionado por los distintos *frameworks* y bibliotecas que se han desarrollado para el uso práctico. Así, un modelo previamente entrenado es una red cuyos parámetros obtenidos se han guardado una vez que se entrenó anteriormente con gran conjunto de datos, o en este caso un gran corpus de texto. De esta forma, se puede usar el modelo preentrenado directamente o adaptarlo a la tarea necesaria para solventar el problema dado. Este proceso de tomar las características aprendidas en un problema y reutilizarlas para otro problema análogo, se denomina aprendizaje por transferencia o, comúnmente conocido por su traducción inglesa, *transfer learning*.

De forma intuitiva, si un modelo es entrenado con un corpus de texto lo suficientemente extenso y genérico, entonces se convierte en un modelo general para la generación de texto y es capaz de realizar diversas tareas genéricas en el campo del *Natural Language Processing*. Además se puede reutilizar la información aprendida para alguna tarea específica sin tener que reentrenar el modelo desde cero.

La secuencia de etapas más común del aprendizaje por transferencia en el contexto del *deep learning* es el siguiente [47]:

1. Se toma el modelo previamente entrenado.
2. Se “congelan” las capas del modelo previamente entrenado para evitar perder información durante el entrenamiento posterior.
3. Se añaden nuevas capas sobre las “congeladas”, cuya finalidad será la de convertir las anteriores características en nuevas predicciones para un conjunto de datos diferente.
4. Se entrena el modelo resultante (concretamente las nuevas capas, ya que las otras están “congeladas”).

Por último, si es necesario, se puede realizar un proceso denominado ajuste fino (o, de nuevo, más comúnmente conocido por su traducción inglesa, *fine-tuning*) que consiste en “descongelar” todo el modelo obtenido (o parte de él) y volver a entrenarlo con los nuevos datos aplicando esta vez una tasa de aprendizaje pequeña. Dependiendo del problema, esto puede lograr mejoras significativas al adaptar gradualmente las funciones previamente entrenadas a los nuevos datos, haciéndolas más relevantes para la tarea específica.

Esta opción será la escogida en uno de los experimentos que se realizarán en el capítulo 5. En él, se elegirá un modelo preentrenado (GPT-2) y se le aplicará la técnica de *fine-tuning* sobre un corpus de discursos de la reina de Inglaterra.

Capítulo 4

Desarrollo matemático del trabajo

Los algoritmos de *machine learning* generalmente requieren una gran cantidad de cálculos numéricos. Para ello se hacen uso de métodos que actualizan las estimaciones de la solución a través de un proceso iterativo, en lugar de derivar analíticamente una fórmula para proporcionar una expresión simbólica para la solución correcta. Las operaciones comunes incluyen la optimización (encontrar el valor de un argumento que minimiza o maximiza una función) y la resolución de sistemas de ecuaciones lineales. Incluso evaluar una función matemática en un ordenador puede ser difícil cuando la función involucra números reales, que no se pueden representar con precisión usando una cantidad finita de memoria.

En esta parte del proyecto se profundizará en los aspectos matemáticos de la optimización de una red neuronal. Los aspectos generales de éstas redes neuronales y su aplicación se encuentra ampliamente explicado en la sección 3.2. Así, de forma complementaria, en el desarrollo de este capítulo, se indagará sobre los aspectos básicos de las funciones de pérdida, su optimización y el entrenamiento de las redes neuronales de forma general, completándolo con las adaptaciones a las redes recurrentes. Además se demostrará la capacidad de dichas arquitecturas neuronales para la resolución del problema que se plantea.

4.1. Notación y consideraciones previas

De aquí en adelante se desarrollará la teoría con puntos de \mathbb{R}^n , que se denotarán con notación en negrita como sigue: $\boldsymbol{x} = [x_1, x_2, \dots, x_n]^\top \in \mathbb{R}^n$.

4.1.1 Definición. Segmento. Sean $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^n$ y $\alpha \in [0, 1]$, entonces el segmento de extremos \boldsymbol{x}_1 y \boldsymbol{x}_2 ($S_{\boldsymbol{x}_1, \boldsymbol{x}_2}$) está formado por los puntos $\boldsymbol{y} \in \mathbb{R}^n$ de la forma $\boldsymbol{y} =$

$\alpha \mathbf{x}_1 + (1 - \alpha) \mathbf{x}_2$, es decir:

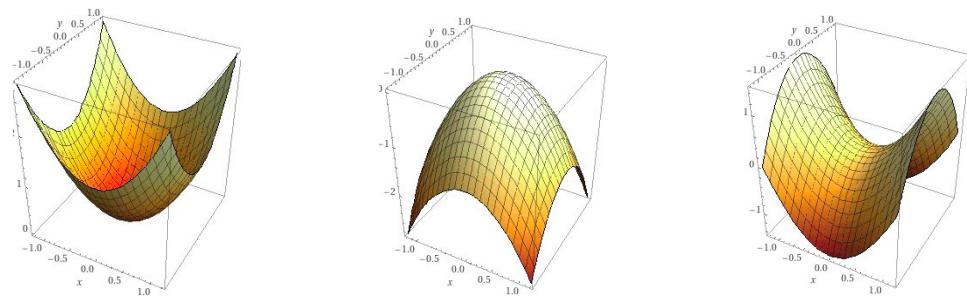
$$\mathcal{S}_{\mathbf{x}_1, \mathbf{x}_2} = \{\mathbf{y} \in \mathbb{R}^n : \mathbf{y} = \alpha \mathbf{x}_1 + (1 - \alpha) \mathbf{x}_2 \ \forall \alpha \in [0, 1]\}$$

4.1.2 Definición. Conjunto convexo. Sea C un conjunto de puntos de \mathbb{R}^n no vacío, se dice que C es un conjunto convexo si para cualquiera dos puntos $\mathbf{x}_1, \mathbf{x}_2 \in C$, el segmento $\mathcal{S}_{\mathbf{x}_1, \mathbf{x}_2} \subset C$.

4.1.3 Definición. Función convexa. Sea $f : C \subset \mathbb{R}^n \rightarrow \mathbb{R}$ una función, se dice que f es una función convexa si C es un conjunto convexo y $\forall \mathbf{x}_1, \mathbf{x}_2 \in C$ y $\forall \alpha \in [0, 1]$ se tiene que $f(\alpha \mathbf{x}_1 + (1 - \alpha) \mathbf{x}_2) \leq \alpha f(\mathbf{x}_1) + (1 - \alpha) f(\mathbf{x}_2)$.

4.1.4 Definición. Función cóncava. Sea $f : C \subset \mathbb{R}^n \rightarrow \mathbb{R}$ una función, se dice que f es una función cóncava si $-f$ es convexa.

A continuación se representan tres funciones, para ver las diferencias de forma visual:



(a) Función convexa (vista desde $(0, 0)$):
 $f(\mathbf{x}) = x_1^2 + x_2^2$

(b) Función cóncava:
 $f(\mathbf{x}) = -(x_1^2 + x_2^2)$

(c) Función ni convexa ni concava: $f(\mathbf{x}) = x_1^2 - x_2^2$

Imagen 4.1: Ejemplos de funciones extraídas de Wolfram|Alpha, donde $\mathbf{x} = [x_1, x_2]^\top \in \mathbb{R}^2$

La mayoría de los algoritmos de *deep learning* hacen uso de métodos de optimización de algún tipo. La optimización se refiere a la tarea de minimizar o maximizar alguna función $f : \mathbb{R}^n \rightarrow \mathbb{R}$ (denominada *función de coste*, *función de pérdida*, entre otros nombres, en la literatura científica) donde a cada $\mathbf{x} \in \mathbb{R}^n$ le hace corresponder un valor $f(\mathbf{x}) \in \mathbb{R}$. Se pretende conseguir esta optimización modificando \mathbf{x} . Por lo general, la mayoría de los problemas de optimización se expresan en términos de minimización de f . La maximización se puede lograr de forma simétrica minimizando $-f$.

Por un lado, la optimización de funciones convexas es el ejemplo más sencillo que se

puede encontrar ya que:

4.1.5 Teorema (Mínimos de funciones convexas). *Sea $f : C \subset \mathbb{R}^n \rightarrow \mathbb{R}$ una función convexa entonces todo mínimo local de $f(C)$ es un mínimo global.*

Demostración

Sea $\mathbf{x}_0 \in C$ un mínimo local, entonces $\exists \epsilon > 0$ tal que $\forall \mathbf{x} \in C$ con $\|\mathbf{x} - \mathbf{x}_0\| < \epsilon$ se tiene que $f(\mathbf{x}_0) \leq f(\mathbf{x})$. Se supone $\exists \mathbf{z} \in C$ tal que $f(\mathbf{z}) < f(\mathbf{x}_0)$. Sea $\alpha \in (0, 1]$ e $\mathbf{y} = \alpha \mathbf{z} + (1 - \alpha) \mathbf{x}_0$, entonces $\mathbf{y} \in C$ por ser C convexo. Además por ser f convexa:

$$f(\mathbf{y}) = f(\alpha \mathbf{z} + (1 - \alpha) \mathbf{x}_0) \leq \alpha f(\mathbf{z}) + (1 - \alpha) f(\mathbf{x}_0) = \alpha(f(\mathbf{z}) - f(\mathbf{x}_0)) + f(\mathbf{x}_0) < f(\mathbf{x}_0)$$

donde el último paso se dio por ser $\alpha > 0$ y $f(\mathbf{z}) - f(\mathbf{x}_0) < 0$ como se ha supuesto.

Así $\forall \mathbf{y} \in S_{\mathbf{z}, \mathbf{x}_0}$, $f(\mathbf{y}) < f(\mathbf{x}_0)$, lo que llega a contradicción ya que tomando un valor de α suficientemente pequeño, se tiene que $\exists \mathbf{y} \in C$ tal que $\|\mathbf{y} - \mathbf{x}_0\| < \epsilon$ con $f(\mathbf{y}) < f(\mathbf{x}_0)$ pero por hipótesis $\forall \mathbf{x} \in C$ con $\|\mathbf{x} - \mathbf{x}_0\| < \epsilon$ entonces $f(\mathbf{x}_0) \leq f(\mathbf{x})$.

De sobra es conocido que para encontrar un mínimo de una función habrá que derivar e igualar a cero. Pues bien, si la función de coste a optimizar es convexa, el mínimo global de la función, si existe, se conseguirá simplemente calculando su derivada.

A continuación se recordaran las nociones básicas de diferenciabilidad en funciones reales de más de una variable:

4.1.6 Definición. Función diferenciable en un punto. *Sea $f : A \subset \mathbb{R}^n \rightarrow \mathbb{R}^m$ una función, $\mathbf{a} \in A^\circ$. Se dice que f es diferenciable en \mathbf{a} si existe una aplicación lineal y continua D verificando:*

$$\lim_{\mathbf{x} \rightarrow \mathbf{a}} \frac{f(\mathbf{x}) - f(\mathbf{a}) - D(\mathbf{x} - \mathbf{a})}{\|\mathbf{x} - \mathbf{a}\|} = 0$$

o análogamente:

$$\lim_{\mathbf{h} \rightarrow 0} \frac{f(\mathbf{a} + \mathbf{h}) - f(\mathbf{a}) - Dh}{\|\mathbf{h}\|} = 0$$

4.1.7 Proposición (Unicidad de la diferencial). *Con las hipótesis de la definición 4.1.6, si f es diferenciable en \mathbf{a} , la aplicación D es única.*

Demostración Como $\mathbf{a} \in A^\circ$ entonces $\exists \delta > 0$ tal que $B(\mathbf{a}, \delta) \subset A$. Para $\mathbf{v} \in \mathbb{R}^n - \{\mathbf{0}\}$, se aplica el cambio de variable $\mathbf{x} = \mathbf{a} + t\mathbf{v}$ con $t \in \mathbb{R}$ y $|t| < \frac{\delta}{\|\mathbf{v}\|}$. Así se tiene, por la linealidad de D que:

$$0 = \lim_{t \rightarrow 0} \frac{\|f(\mathbf{a} + t\mathbf{v}) - f(\mathbf{a}) - t(D\mathbf{v})\|}{|t|\|\mathbf{v}\|} = \frac{1}{\|\mathbf{v}\|} \lim_{t \rightarrow 0} \left\| \frac{f(\mathbf{a} + t\mathbf{v}) - f(\mathbf{a})}{t} - D\mathbf{v} \right\|$$

Por tanto:

$$D\mathbf{v} = \lim_{t \rightarrow 0} \frac{f(\mathbf{a} + t\mathbf{v}) - f(\mathbf{a})}{t}$$

Como esto es válido para todo $\mathbf{v} \in \mathbb{R}^n - \{\mathbf{0}\}$, entonces D es única y se denomina la diferencial de f en \mathbf{a} , denotada como $Df(\mathbf{a})$.

4.1.8 Definición. Sea Ω un subconjunto abierto de \mathbb{R}^n y $f : \Omega \rightarrow \mathbb{R}^m$ una función diferenciable para todo $\omega \in \Omega$, entonces se dirá que f es diferenciable y a la función, Df , que a cada punto $\omega \in \Omega$ le hace corresponder la diferencial $Df(\omega)$ se le denominará la diferencial de f .

4.1.9 Teorema (Regla de la cadena). Sean \mathcal{A}, \mathcal{B} abiertos, no vacíos, de R^n y R^m respectivamente y $f : \mathcal{A} \rightarrow \mathcal{B}$ y $g : \mathcal{B} \rightarrow \mathbb{R}^t$ funciones. Si f es diferenciable en un punto $\mathbf{a} \in \mathcal{A}$ y g es diferenciable en un punto $\mathbf{b} = f(\mathbf{a}) \in \mathcal{B}$, entonces $g \circ f$ es diferenciable en \mathbf{a} , con:

$$D(g \circ f)(\mathbf{a}) = Dg(f(\mathbf{a})) \circ Df(\mathbf{a})$$

4.1.10 Definición. Derivada de una función en una dirección. Sea Ω un subconjunto abierto de \mathbb{R}^n , $f : \Omega \rightarrow \mathbb{R}^m$ una función, $\mathbf{a} \in \Omega$, $\mathbf{u} \in \mathbb{R}^n$ con $\|\mathbf{u}\| = 1$ y el límite $\lim_{t \rightarrow 0} \frac{f(\mathbf{a} + t\mathbf{u}) - f(\mathbf{a})}{t}$ existe, entonces la derivada de f en \mathbf{a} en la dirección \mathbf{u} se define como:

$$D_{\mathbf{u}}f(\mathbf{a}) = \lim_{t \rightarrow 0} \frac{f(\mathbf{a} + t\mathbf{u}) - f(\mathbf{a})}{t}$$

Además si el vector \mathbf{u} se corresponde con el vector \mathbf{e}_i de la base canónica de \mathbb{R}^n para cualquier $i \in \{1, \dots, n\}$, se denomina a dicha derivada como la **derivada parcial de f en v con respecto a la variable x_i** en el punto \mathbf{a} y se denota como $\frac{\partial f}{\partial x_i}(\mathbf{a})$.

4.1.11 Definición. Gradiente de una función en un punto. Sea Ω un subconjunto abierto de \mathbb{R}^n , $f : \Omega \rightarrow \mathbb{R}$ una función, $\mathbf{a} \in \Omega$. Si existen $\frac{\partial f}{\partial x_i}(\mathbf{a}) \forall i \in \{1, \dots, n\}$, entonces el gradiente de f en \mathbf{a} se define como:

$$\nabla f(\mathbf{a}) = \left(\frac{\partial f}{\partial x_1}(\mathbf{a}), \frac{\partial f}{\partial x_2}(\mathbf{a}), \dots, \frac{\partial f}{\partial x_n}(\mathbf{a}) \right)$$

En un contexto dado por Ω un subconjunto abierto de \mathbb{R}^n , $\mathbf{a} \in \Omega$ y $f : \Omega \rightarrow \mathbb{R}$ una función para la que existen todas las derivadas direccionales posibles en \mathbf{a} .

Fijado un punto $\mathbf{u} \in \mathbb{R}^n$ con $\|\mathbf{u}\| = 1$, el desplazamiento de una distancia $t > 0$ desde el punto \mathbf{a} , en la dirección y sentido de \mathbf{u} , el movimiento viene dado por $f(\mathbf{a} + t\mathbf{u}) - f(\mathbf{a})$. Si f es diferenciable en \mathbf{a} , entonces $D_{\mathbf{u}}f(\mathbf{a}) = (\nabla f(\mathbf{a})|\mathbf{u}) \quad \forall \mathbf{u} \in \mathbb{R}^n$ con $\|\mathbf{u}\| = 1$. Si $\nabla f(\mathbf{a}) \neq \mathbf{0}$, tomando $\mathbf{v} = \frac{\nabla f(\mathbf{a})}{\|\nabla f(\mathbf{a})\|}$, por la desigualdad de Cauchy-Schwartz:

$$D_{\mathbf{u}}f(\mathbf{a}) = (\nabla f(\mathbf{a})|\mathbf{u}) \leq \|\nabla f(\mathbf{a})\| = (\nabla f(\mathbf{a})|\mathbf{v}) = D_{\mathbf{v}}f(\mathbf{a})$$

Por tanto se deduce que:

$$D_{\mathbf{v}}f(\mathbf{a}) = \max \{D_{\mathbf{u}}f(\mathbf{a}) : \mathbf{u} \in \mathbb{R}^n, \|\mathbf{u}\| = 1\} > 0$$

Así por tanto, el vector gradiente normalizado ($\mathbf{v} = \frac{\nabla f(\mathbf{a})}{\|\nabla f(\mathbf{a})\|}$) es la única dirección que hace que la derivada direccional, $D_{\mathbf{v}}f(\mathbf{a})$, sea máxima.

Por tanto, al realizar pequeñas variaciones desde el punto \mathbf{a} en la dirección y el sentido del vector gradiente, se consigue la máxima tasa de aumento de f . Si el sentido es el inverso, se consigue la máxima tasa de disminución de f . Esta será la idea clave que se desarrollará en el método del Gradiente Descendente (sección 4.3.1).

4.2. Funciones de pérdida

En esta sección se profundizará sobre las funciones de pérdida más usadas en el campo del *machine learning*. Se agrupan según el problema que se tiene entre manos: regresión, clasificación binaria y clasificación multiclas. Realmente muchas funciones de pérdida se pueden adaptar tanto para regresión como para clasificación, sin embargo, a continuación, se van a agrupar dentro del grupo de problemas donde tiene un uso más común.

Antes de avanzar es importante hacer una aclaración: la diferencia entre una función de error y una función de pérdida. Aunque la mayoría de veces se usan indistintamente, una función de error mide la desviación de un valor observable respecto a una predicción, por tanto es un objeto puramente estadístico; mientras que una función de pérdida opera sobre dicho error para cuantificar la consecuencia negativa de este y por tanto es un objeto asociado a la teoría de la decisión.

Por ejemplo, para diversos problemas es útil usar una pérdida de error al cuadrado, donde la consecuencia negativa de dicho error se cuantifica como proporcional al cuadrado de este. Para otros problemas, puede ser útil valorar la influencia del error en una dirección particular (por ejemplo, preferencia de un falso positivo frente a falso negativo) y, por lo tanto, podríamos adoptar una función de pérdida no simétrica.

A continuación se muestran algunas de las funciones de pérdida más usadas, aunque existen infinidad de variantes.

4.2.1. Funciones de pérdida para problemas de regresión

Un problema de modelado predictivo de regresión se basa en calcular la predicción de una cantidad con valor real.

Error cuadrático medio (MSE)

El error cuadrático medio (*Mean Squared Error*, MSE) es una función de pérdida que mide el promedio de las diferencias cuadráticas entre los valores predichos y los reales.

Dados N datos de entrenamiento usados para estimar el error, J el error cuadrático medio, $\hat{\mathbf{y}}_i$ el vector que el modelo ha predicho para el dato i , \mathbf{y}_i el verdadero vector observado para el dato i (con $i \in \{1, \dots, N\}$), entonces:

$$J = \frac{1}{N} \sum_{i=1}^N \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_2^2$$

De esta forma el resultado es siempre un número positivo o 0 y su óptimo es este último. Al elevar al cuadrado se introduce una penalización en aquellas diferencias más elevadas, es decir, se penaliza los errores más grandes frente a los más pequeños.

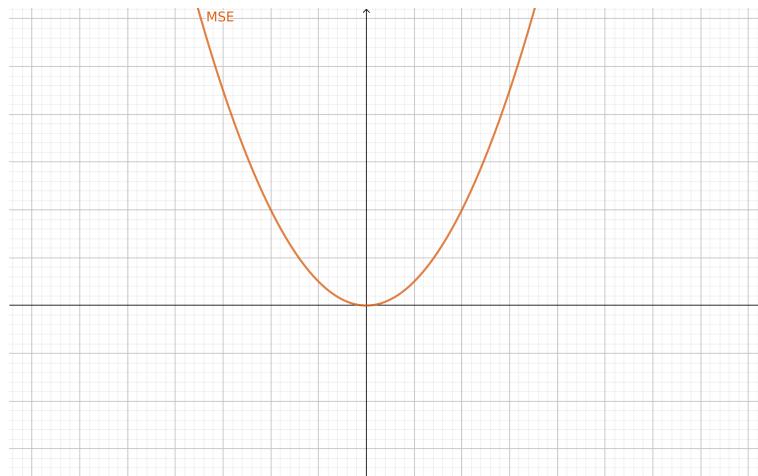


Imagen 4.2: Función de pérdida MSE con $N = 1$.

Error medio absoluto (MAE)

El error medio absoluto (*Mean Absolute Error*, MAE) es una función de pérdida que mide promedio de las diferencias absolutas entre los valores predichos y los reales.

Dados N datos de entrenamiento usados para estimar el error, J el error medio absoluto, $\hat{\mathbf{y}}_i$ el vector que el modelo ha predicho para el dato i , \mathbf{y}_i el verdadero vector observado para el dato i (con $i \in \{1, \dots, N\}$), entonces:

$$J = \frac{1}{N} \sum_{i=1}^N \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_1$$

De nuevo, el resultado es siempre un número positivo o 0 y su óptimo es el 0. Además es más robusto que MSE frente a datos atípicos u *outliers* ya que no dispara el valor del error.

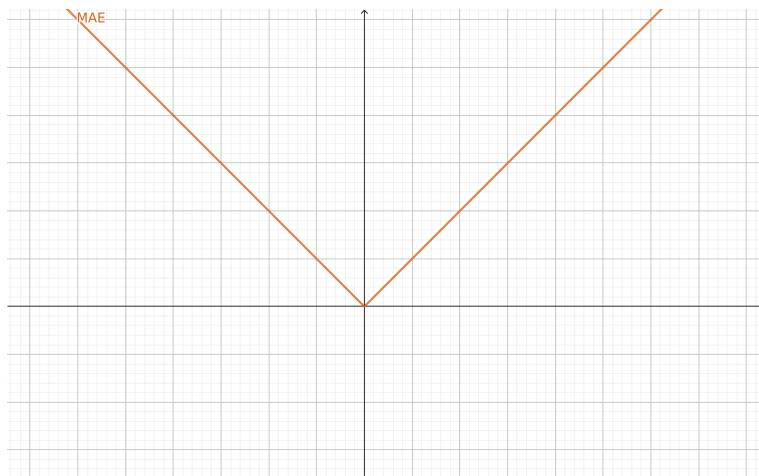


Imagen 4.3: Función de pérdida MAE con $N = 1$.

Coefficiente de error R^2

El coeficiente de error R^2 es una función de pérdida que hace uso del error cuadrático medio pero está libre de escala.

Dados $N > 1$ datos de entrenamiento distintos usados para estimar el error, J el coeficiente de error R^2 , $\hat{\mathbf{y}}_i$ el vector que el modelo ha predicho para el dato i , \mathbf{y}_i el verdadero vector observado para el dato i e $\bar{\mathbf{y}}$ la media de los verdaderos valores de los datos de entrenamiento usados para estimar el error (con $i \in \{1, \dots, N\}$), entonces:

$$J = 1 - \frac{\frac{1}{N} \sum_{i=1}^N \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_2^2}{\frac{1}{N} \sum_{i=1}^N \|\mathbf{y}_i - \bar{\mathbf{y}}\|_2^2}$$

En este caso el resultado es un valor perteneciente al intervalo $(-\infty, 1]$, su óptimo es el 1 y si es negativo, el modelo predicho es peor que la media. Al estar libre de escala es mucho más fácil de interpretar que las otras métricas.

Función de pérdida de Huber

El error de Huber es una función de pérdida que combina MSE y MAE. Intuitivamente es como el error medio absoluto pero que adopta la forma cuadrática cuando el valor es pequeño, proporcionando derivabilidad a la función. Dicho margen lo determina el hiperparámetro $\delta \in \mathbb{R}^+$.

Dados N datos de entrenamiento usados para estimar el error, J_δ la función de pérdida de Huber dado un $\delta \in \mathbb{R}^+$ determinado, $\hat{\mathbf{y}}_i$ el vector que el modelo ha predicho para el dato i y \mathbf{y}_i el verdadero vector observado para el dato i (con $i \in \{1, \dots, N\}$), entonces:

$$J_\delta = \frac{1}{N} \sum_{i=1}^N \begin{cases} \frac{1}{2} \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_2^2 & \text{si } \|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_1 \leq \delta \\ \delta (\|\mathbf{y}_i - \hat{\mathbf{y}}_i\|_1 - \frac{1}{2}\delta) & \text{en otro caso} \end{cases}$$

De nuevo, el resultado es siempre un número positivo o 0 y su óptimo es el 0. Esta función de pérdida amortigua el gradiente elevado que sí provoca MAE al aplicarse como función de pérdida en los entrenamientos de redes neuronales y además es más robusta a valores atípicos que la función de pérdida MSE. El inconveniente es que hay que estimar un valor del hiperparámetro δ .

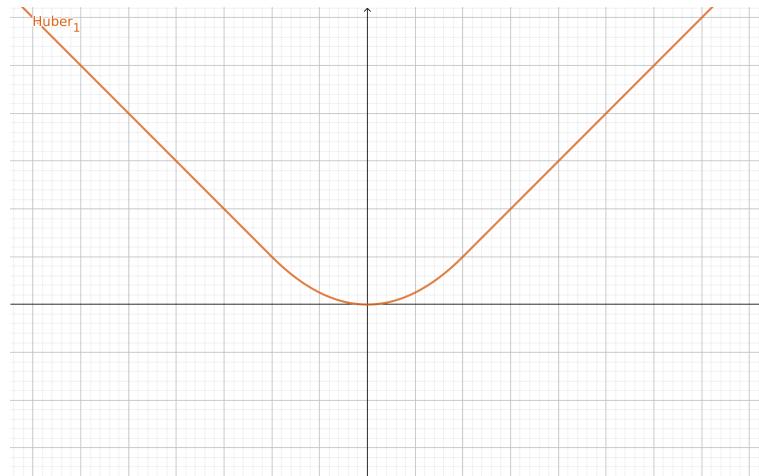


Imagen 4.4: Función de pérdida de Huber con $\delta = 1$ y $N = 1$.

Función de pérdida Log-Cosh

La función de pérdida de Log-Cosh es muy parecida a la función de Huber. Se basa en el logaritmo del coseno hiperbólico del error de predicción. Suaviza la función MSE pero a la vez es derivable.

Dados N datos de entrenamiento usados para estimar el error, J la función de pérdida Log-Cosh, $\hat{\mathbf{y}}_i$ el vector de tamaño n que el modelo ha predicho para el dato i donde la

componente j se denota por $\hat{y}_{i,j}$ e \mathbf{y}_i el verdadero vector observado para el dato i donde la componente j se denota por $y_{i,j}$ (con $i \in \{1, \dots, N\}$), entonces:

$$J = \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^n \log(\cosh(y_{i,j} - \hat{y}_{i,j}))$$

Otra vez, el resultado es siempre un número positivo o 0 y su óptimo es el 0. Evita el tener que estimar un hiperparámetro y además es dos veces diferenciable.

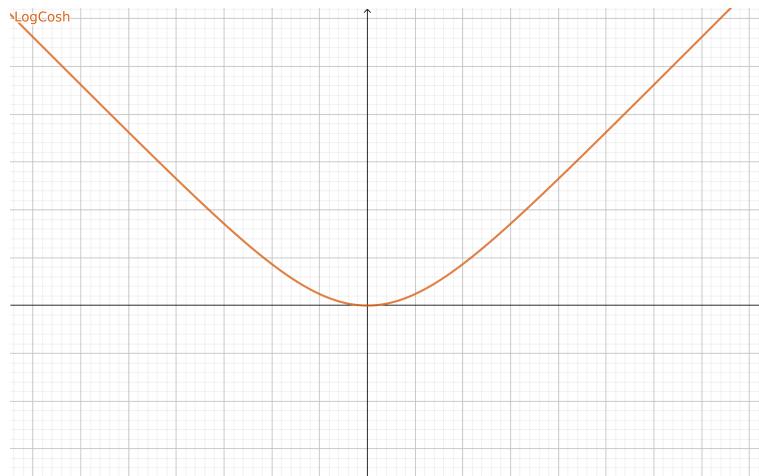


Imagen 4.5: Función de pérdida Log-Cosh con $N = 1$ y $n = 1$.

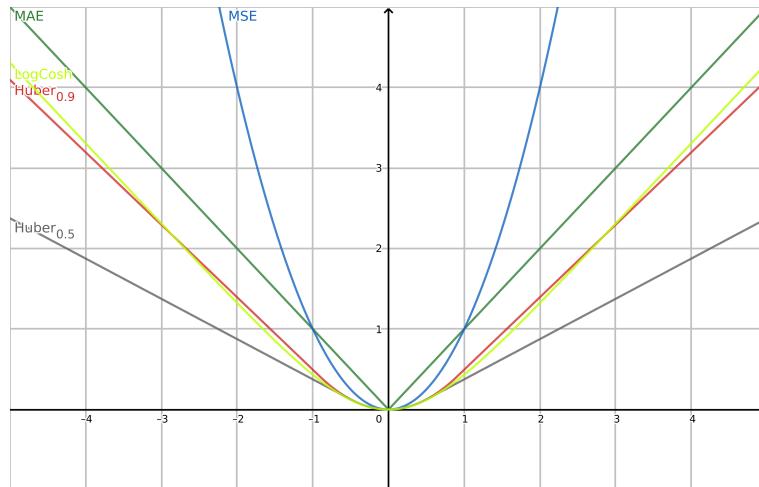


Imagen 4.6: Comparativa entre las distintas funciones de pérdida típicas estudiadas para regresión: MAE, MSE, Huber ($\delta = 0,9$), Huber ($\delta = 0,5$) y Log-Cosh.

4.2.2. Funciones de pérdida para problemas de clasificación binaria

Un problema de modelado predictivo de clasificación binaria se basa en calcular la predicción de una cantidad con valor binario (cada uno de estos valores representa un

grupo o una clase).

Entropía cruzada (binaria)

La función de pérdida de entropía cruzada (*cross-entropy*) es una función de pérdida que calcula una puntuación que indica la diferencia promedio entre las distribuciones de probabilidad reales y predichas para las dos clases.

Dados N datos de entrenamiento usados para estimar el error, J la entropía cruzada, $\hat{y}_i \in (0, 1)$ la probabilidad de que pertenezca a la clase que el modelo ha predicho para el dato i , y_i el verdadero valor para el dato i (con $i \in \{1, \dots, N\}$) y suponiendo $y_i \in \{0, 1\}$, entonces:

$$J = -\frac{1}{N} \sum_{i=1}^N y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$$

Aplicando la fórmula, el resultado es siempre un número positivo o 0 y su óptimo es el 0. Dicha función de pérdida penaliza de forma muy elevada los errores en la predicción.

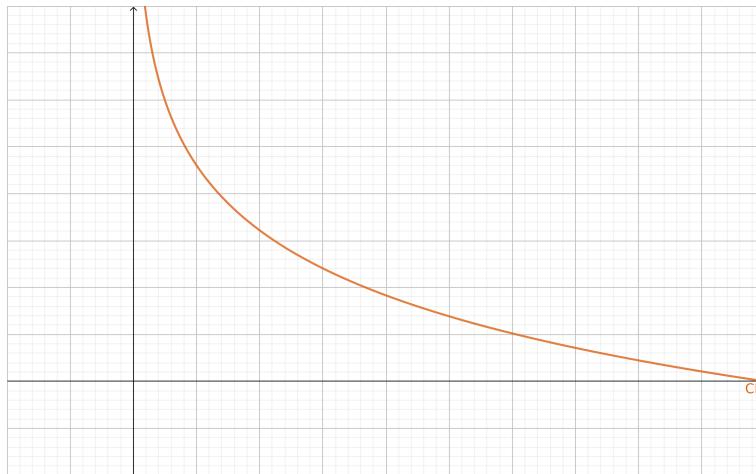


Imagen 4.7: Función de pérdida entropía cruzada cuando $y_i = 1$ y $N = 1$.

Función de pérdida de Hinge

La función de pérdida de Hinge es una función de pérdida diseñada para encontrar la frontera de decisión lo más distante posible de cada ejemplo de entrenamiento, por eso es la que más se usa en las máquinas de vectores de soporte (SVM).

Dados N datos de entrenamiento usados para estimar el error, J la función de pérdida de Hinge, $\hat{y}_i \in \mathbb{R}$ el valor que el modelo ha predicho para el dato i , y_i el verdadero valor para el dato i (con $i \in \{1, \dots, N\}$) y suponiendo $y_i \in \{-1, 1\}$, entonces:

$$J = \sum_{i=1}^N \max(0, 1 - y_i \hat{y}_i)$$

El resultado es siempre un número positivo o 0 y su óptimo es el 0.

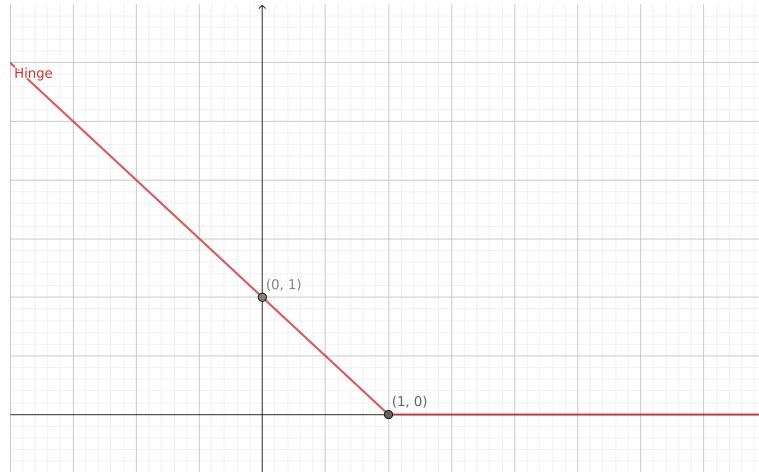


Imagen 4.8: Función de pérdida Hinge con $y_i = 1$ y $N = 1$.

Función de pérdida de Hinge cuadrática

La función de pérdida de Hinge cuadrática es una función de pérdida análoga a la función de Hinge, pero incluyendo una potencia cuadrática para penalizar de mayor medida los valores atípicos.

Dados N datos de entrenamiento usados para estimar el error, J la función de pérdida de Hinge cuadrática, $\hat{y}_i \in \mathbb{R}$ el valor que el modelo ha predicho para el dato i , y_i el verdadero valor para el dato i (con $i \in \{1, \dots, N\}$) y suponiendo $y_i \in \{-1, 1\}$, entonces:

$$J = \sum_{i=1}^N \max(0, 1 - y_i \hat{y}_i)^2$$

El resultado es siempre un número positivo o 0 y su óptimo es el 0.

4.2.3. Funciones de pérdida para problemas de clasificación multiclase

Un problema de modelado predictivo de clasificación se basa en calcular la predicción de una cantidad con valor discreto mayor que 2 (cada uno de estos valores representa un grupo o una clase).

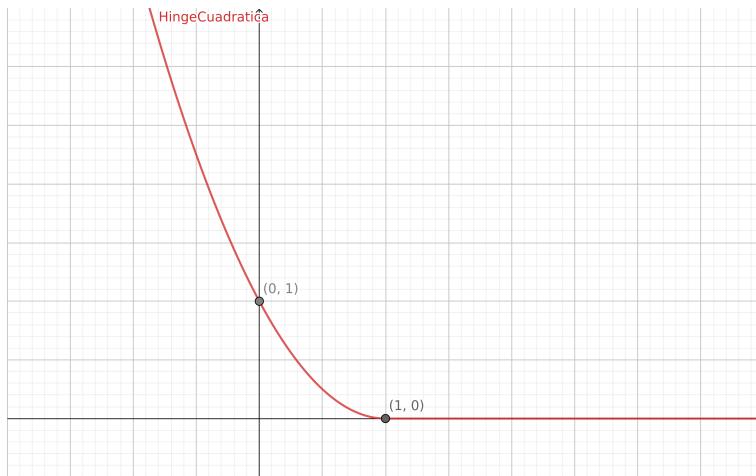


Imagen 4.9: Función de pérdida Hinge cuadrática con $y_i = 1$ y $N = 1$.

Entropía cruzada (multiclasificación)

La función de pérdida de entropía cruzada adaptada a múltiples clases (*Multi-Class Cross-Entropy*) es una función de pérdida que calcula una puntuación que indica la diferencia promedio entre las distribuciones de probabilidad reales y predichas para las distintas clases del problema.

Dados N datos de entrenamiento usados para estimar el error, n clases, J la entropía cruzada, $\hat{\mathbf{y}}_i \in (0, 1)^n$ la distribución de probabilidad de pertenencia a cada clase que el modelo ha predicho para el dato i donde $\hat{y}_{i,j}$ es la componente j del vector que indica la probabilidad predicha de pertenecer a la clase j ($j \in \{1, \dots, n\}$), \mathbf{y}_i la distribución de probabilidad que modela el verdadero valor para el dato i (con $i \in \{1, \dots, N\}$) donde $y_{i,j}$ es la componente j del vector que indica la clase j verdadera ($j \in \{1, \dots, n\}$) y suponiendo $\mathbf{y}_i \in \{0, 1\}^n$ con un 1 en la posición que indica la clase que pertenece, entonces:

$$J = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^n y_{i,j} \log(\hat{y}_{i,j})$$

Aplicando la fórmula, el resultado es siempre un número positivo o 0 y su óptimo es el 0. Dicha función de pérdida penaliza de forma muy elevada los errores en la predicción.

Divergencia de Kullback-Leibler

La función de pérdida conocida como divergencia de Kullback-Leibler mide la diferencia entre una distribución de probabilidad y otra distribución base.

Dados N datos de entrenamiento usados para estimar el error, n clases, J la divergencia de Kullback-Leibler, $\hat{\mathbf{y}}_i \in (0, 1)^n$ la distribución de probabilidad de pertenencia a cada

clase que el modelo ha predicho para el dato i donde $\hat{y}_{i,j}$ es la componente j del vector que indica la probabilidad predicha de pertenecer a la clase j ($j \in \{1, \dots, n\}$), $\mathbf{y}_i \in (0, 1)^n$ una distribución de probabilidad de pertenencia a cada clase observada para un dato i (con $i \in \{1, \dots, N\}$) donde $y_{i,j}$ es la componente j del vector que indica la probabilidad de la clase j ($j \in \{1, \dots, n\}$), entonces:

$$J = -\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^n y_{i,j} (\log(y_{i,j}) - \log(\hat{y}_{i,j}))$$

También se puede suponer que $\mathbf{y}_i \in \{0, 1\}^n$ con un 1 en la posición que indica la clase que pertenece, lo que también forma una distribución de probabilidad, sin embargo, en este caso, computacionalmente, la divergencia de Kullback-Leiber se transforma en la entropía cruzada.

Así, un valor de pérdida de Kullback-Leiber de 0 indica que las distribuciones son idénticas. Intuitivamente y de forma análoga a la entropía cruzada, se calcula cuánta información se pierde si la distribución de probabilidad predicha se usa para aproximar la distribución de probabilidad objetivo deseada.

4.3. Métodos de optimización

Ya se ha explicado como conseguir una optimización de la función de coste de funciones convexas, pero en la práctica pueden surgir varios problemas con las funciones de coste: el más obvio, que la función no sea convexa, lo que puede provocar que existan una gran cantidad de mínimos locales y por tanto gran cantidad de ecuaciones que resolver o la optimización se puede estancar en alguno de ellos, bastante lejos del mínimo global. Se tratarán de solucionar dichos problemas con los métodos que se desarrollan a continuación.

4.3.1. Gradiente descendente

La gran variedad de modelos y funciones de coste en el campo de *machine learning* ha obligado a encontrar soluciones para optimizar todo tipo de funciones, tanto convexas como no-convexas.

El descenso de gradiente es una método iterativo general para minimizar una función de coste $f : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ diferenciable, en función de sus parámetros $\boldsymbol{\omega} = [\omega_0, \omega_1, \dots, \omega_n] \in \mathbb{R}^{n+1}$.

Se considera también que cuando se inicializa el entrenamiento, los parámetros a optimizar ($\boldsymbol{\omega}$) se inicializan con valores aleatorios ($\boldsymbol{\omega}^{(0)} \in \mathbb{R}^{n+1}$).

Como la función f es diferenciable en el punto $\omega^{(0)}$ (4.1.6), se puede calcular el gradiente de la función f en dicho punto (4.1.11), y como se explicó anteriormente, si se modifica el punto inicial $\omega^{(0)}$ en la dirección del gradiente normalizado ($v = \frac{\nabla f(a)}{\|\nabla f(a)\|}$) con sentido opuesto al vector gradiente, se obtiene la máxima tasa de disminución de f en ese punto.

$$\omega^{(1)} = \omega^{(0)} - \nabla f(\omega^{(0)})$$

Así pues, al realizar esta operación se obtendrá una actualización de los parámetros. Si se itera con esta ecuación se obtiene una sucesión de vectores $\{\omega^{(0)}, \omega^{(1)}, \dots, \omega^{(m)}, \dots\}$ parámetros de nuestro modelo que converge a un mínimo local de la función f .

Sin embargo, el método presenta algunas lagunas, ya que siempre se mueve una unidad en el sentido contrario al gradiente. Pero la cuestión es, ¿cuánta distancia es la correcta para desplazarse?, o dicho de otra forma, si se incluye un parámetro $\alpha \in \mathbb{R}^+$, denominado ratio o tasa de aprendizaje (*learning rate*, en inglés) en la ecuación anterior, de tal forma que resulte: $\omega^{(1)} = \omega^{(0)} - \alpha \nabla f(\omega^{(0)})$ ¿con qué valor α se obtendrá una mejor optimización de la función en una iteración del método?.

Pues bien, el valor de α óptimo no se conoce a priori para un problema dado. Para obtenerlo, se suelen usar los valores que anteriormente se han aplicado en problemas similares obteniendo buenos resultados o se buscan mediante prueba y error. Una buena forma de obtener el valor óptimo de la tasa de aprendizaje, como cualquier hiperparámetro, será aplicar una búsqueda aleatoria o dirigida en rejilla de parámetros o aplicar una optimización bayesiana.

La figura 4.10 ilustra gráficamente esta idea: si la tasa de aprendizaje es demasiado pequeña, aunque la sucesión de parámetros obtenida se acerca al mínimo, se realizan una gran cantidad de iteraciones, lo que provoca una lenta convergencia y un tiempo de ejecución muy elevado. Además puede provocar que converja con mayor probabilidad en un mínimo local más ineficiente. Por el contrario, si la tasa de aprendizaje es demasiado elevada, la sucesión puede llegar a no converger.

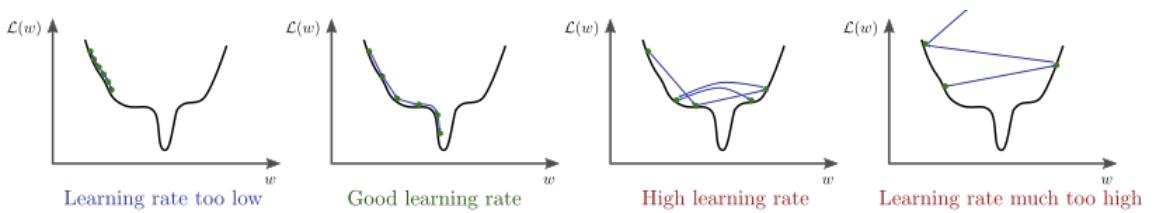


Imagen 4.10: Ejemplos de diferentes valores de tasa de aprendizaje escogida para la aplicación del gradiente descendente a una función dada [44].

Este es el método del gradiente descendente y se le atribuye a Cauchy en el año 1847. Formalmente:

$$\omega^{(i+1)} = \omega^{(i)} - \alpha \nabla f(\omega^{(i)}) \quad (4.1)$$

Hay diferentes mejoras de este método para que la variación del punto inicial sea más eficiente con respecto a la optimización. Sin embargo, este método es la técnica básica de optimización de redes neuronales.

4.3.2. Gradiente descendente estocástico (SGD)

La versión de descenso de gradiente descrita en la anterior sección se conoce como descenso de gradiente por lotes (*batch gradient descent*) debido a que el gradiente se calcula para el error en todo el conjunto de datos de entrenamiento antes de que se realice una actualización de los parámetros.

En la práctica resulta más eficiente una versión de dicho algoritmo: el descenso de gradiente estocástico (*stochastic gradient descent*) (SGD). En lugar de considerar el gradiente de lote completo en todos los N datos de entrenamiento, se considera una versión estocástica del gradiente.

De nuevo, se supone la función de coste $f : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ diferenciable, en función de sus parámetros $\omega = [\omega_0, \omega_1, \dots, \omega_n] \in \mathbb{R}^{n+1}$, que en el primer paso se inicializarán con valores aleatorios $(\omega^{(0)} \in \mathbb{R}^{n+1})$.

Así, el método iterativo consistirá en elegir un punto del conjunto de datos de entrenamiento (x_j, Y_j) donde x_j es el punto e Y_j el valor de salida para dicho dato, con $j \in \{1, \dots, N\}$. Este punto se escoge siguiendo una distribución aleatoria uniforme, y se considera el error solo en dicho punto $(f(\omega^{(0)}))_{(x_j, Y_j)}$.

El gradiente del error de este único punto de datos $\nabla f(\omega^{(0)})_{(x_j, Y_j)}$ se usa para la actualización de parámetros exactamente de la misma manera que se usó el gradiente en el descenso de gradiente por lotes (eq. 4.1). Es decir, dado un $\alpha > 0$ (*learning rate*), una iteración $i \geq 0$ y un punto escogido de forma uniformemente aleatoria (x_j, Y_j) con $j \in \{1, \dots, N\}$, la actualización de los parámetros, con respecto a la función de pérdida f , es la siguiente:

$$\omega^{(i+1)} = \omega^{(i)} - \alpha \nabla(f(\omega^{(i)}))_{(x_j, Y_j)} \quad (4.2)$$

Si se calcula el valor esperado del cambio de parámetros con respecto al punto aleatorio escogido, como j se escoge de forma uniformemente aleatoria entre $\{1, \dots, N\}$, el cambio de parámetros esperado es:

$$-\alpha \frac{1}{N} \sum_{i=1}^N \nabla(f(\omega^{(i)}))_{(\mathbf{x}_j, Y_j)}$$

que es exactamente el mismo que la variación de parámetros determinista de descenso del gradiente. Es decir, en promedio, la minimización avanza en la dirección correcta, pero es un poco oscilante, aunque, a la larga, las oscilaciones aleatorias se anulan. Sin embargo, el costo computacional al calcular el gradiente para solo un punto por iteración es más eficiente que calcularlo para todos los N puntos como lo se realiza en el descenso de gradiente.

Una alternativa, entre medias, del gradiente descendente original y el gradiente descendente estocástico original es calcular el gradiente frente a más de un ejemplo de entrenamiento (denominado *mini-batch*) en cada iteración. Se ha demostrado que puede funcionar significativamente mejor que el descenso de gradiente estocástico original gracias a las bibliotecas de vectorización y a la convergencia más suave que surge al promediar el gradiente con más muestras de entrenamiento.

4.3.3. Momentum

Momentum [35] es una técnica de optimización que amortigua las oscilaciones que produce la técnica de SGD e intenta acelerar su convergencia. Para conseguirlo añade una fracción γ del vector de actualización de la iteración anterior $i - 1$, \mathbf{v}_{i-1} , al vector de actualización actual i , \mathbf{v}_i .

$$\begin{aligned}\mathbf{v}_i &= \gamma \mathbf{v}_{i-1} + \alpha \nabla f(\omega^{(i-1)}) \\ \omega^{(i)} &= \omega^{(i-1)} - \mathbf{v}_i\end{aligned}$$

La idea intuitiva es incluir inercia en la optimización, es decir, el término de actualización aumenta para aquellas dimensiones cuyos gradientes apuntan en las mismas direcciones y reduce el término de actualización para las dimensiones cuyos gradientes cambian de dirección. Como resultado, se obtiene una convergencia más rápida y se reduce la oscilación.

4.3.4. Descenso acelerado de Nesterov (NAG)

El descenso acelerado de Nesterov (*Nesterov accelerated gradient*, NAG) [36] introduce una mejora al método de Momentum, ya que dicha técnica no evita el *overshooting*, es decir, sobrepasa el punto mínimo y no converge, que es lo que ocurre en la tercera y cuarta figura de la imagen 4.10.

El algoritmo calcula el desplazamiento antes de aplicarlo, evalúa el gradiente en dicho punto y realiza una corrección de dicho desplazamiento al actualizar los parámetros. Con dicho método se tiene una visión futura del resultado de la actualización antes de realizarla.

$$\begin{aligned}\mathbf{v}_i &= \gamma \mathbf{v}_{i-1} + \alpha \nabla f(\boldsymbol{\omega}^{(i-1)} - \gamma \mathbf{v}_{i-1}) \\ \boldsymbol{\omega}^{(i)} &= \boldsymbol{\omega}^{(i-1)} - \mathbf{v}_i\end{aligned}$$

Al evitar el *overshooting*, el descenso acelerado de Nesterov permite una mayor rapidez de convergencia que los demás métodos estudiados hasta ahora, en general.

4.3.5. Descenso de Gradiente Adaptable (Adagrad)

El descenso del gradiente adaptable (*Adaptive Gradient Algorithm*, Adagrad) [37], permite adaptar las actualizaciones de los parámetros individualmente a cada variable para realizar actualizaciones más grandes o más pequeñas según su frecuencia, por lo que es una técnica adecuada para tratar la escasez de datos, como por ejemplo, en el NLP ya que se trata con palabras poco frecuentes, que necesitarán actualizaciones mayores, frente a otras más frecuentes.

Manteniendo la notación y usando el símbolo $g_i^{(t)}$ para denotar al gradiente de la función de coste f con respecto al parámetro w_i en la t -ésima iteración ($w_i^{(t)}$), se presenta la actualización por parámetro realizada por el algoritmo:

$$g_i^{(t)} = \nabla[f(\boldsymbol{\omega}^{(t)})]_i$$

Además, se modifica la tasa de aprendizaje (*learning rate*) en cada iteración t según los gradientes $g_i^{(k)}$ anteriores ($k \leq t$), en concreto se usa la suma al cuadrado de los gradientes anteriores, es decir, $G_i^{(t)} = \sum_{k=1}^t (g_i^{(k)})^2$. Con dicha notación, la actualización de cada parámetro en cada iteración se calcula como:

$$\omega_i^{(t+1)} = \omega_i^{(t)} - \frac{\alpha}{\sqrt{G_i^{(t)} + \epsilon}} g_i^{(t)}$$

donde $\epsilon \in \mathbb{R}^+$ es un término que evita la división entre cero. Ahora, si se denota por $\mathbf{G}^{(t)} \in \mathbb{R}^{n+1}$, para cada iteración t , al vector que tiene en cada posición i el valor $G_i^{(t)}$ calculado anteriormente, se obtiene que la actualización de parámetros en cada iteración del algoritmo, de forma compacta, queda de la siguiente forma:

$$\boldsymbol{\omega}^{(t+1)} = \boldsymbol{\omega}^{(t)} - \frac{\alpha}{\sqrt{\mathbf{G}^{(t)} + \epsilon}} \otimes \mathbf{g}^{(t)}$$

donde \otimes es la multiplicación elemento a elemento, $\mathbf{g}^{(t)}$ el gradiente de la función de coste f con respecto a todos los parámetros y todas las demás operaciones son elemento a elemento.

De esta forma, Adagrad evita el ajuste manual del hiperparámetro α o *learning rate*, ya que se autoajusta en el desarrollo de la optimización. Sin embargo, al acumular gradientes cuadrados en el denominador durante el entrenamiento, la tasa de aprendizaje se reduce y se vuelve insignificante, por lo que el algoritmo puede estancarse al ajustar los parámetros con el paso de las iteraciones.

4.3.6. Adadelta

La técnica Adadelta [38] es una mejora de Adagrad que surge motivada por el problema de reducción de la tasa de aprendizaje con el paso de las iteraciones. Así, Adadelta restringe la cantidad acumulada de gradientes anteriores a la actual iteración, a un tamaño fijo, en vez de ir acumulando todos los gradientes anteriores.

Para ello, en lugar de almacenar de manera ineficiente los gradientes cuadrados anteriores, calcula la suma de los gradientes de forma recursiva como un promedio decreciente de todos los gradientes cuadrados anteriores, $\mathbf{E}[\mathbf{g}^2]^{(t)} \in \mathbb{R}^{n+1}$:

$$\mathbf{E}[\mathbf{g}^2]^{(t)} = \gamma \mathbf{E}[\mathbf{g}^2]^{(t-1)} + (1 - \gamma) \mathbf{g}^{(t)2}$$

con $\gamma \in (0, 1)$. Así, la variación de los parámetros se calcula como:

$$\Delta\omega^{(t)} = -\frac{\alpha}{\sqrt{\mathbf{E}[\mathbf{g}^2]^{(t)} + \epsilon}} \otimes \mathbf{g}^{(t)}$$

Simplificando la notación, se puede apreciar que el denominador es el criterio de error de la raíz de la media cuadrática (*root mean square*, RMS) del gradiente, con una constante de suavización. Generalizando y llamando al denominador $\text{RMS}[\mathbf{g}]^{(t)} = \sqrt{\mathbf{E}[\mathbf{g}^2]^{(t)} + \epsilon}$, se tiene que:

$$\Delta\omega^{(t)} = -\frac{\alpha}{\text{RMS}[\mathbf{g}]^{(t)}} \otimes \mathbf{g}^{(t)}$$

Por otro lado también se hace un cambio en el denominador, para hacer coincidir las unidades de la actualización con la de los parámetros. Para ello se utiliza la media:

$$\mathbf{E}[\Delta\omega^2]^{(t)} = \gamma \mathbf{E}[\Delta\omega^2]^{(t-1)} + (1 - \gamma) \Delta\omega^2$$

Siguiendo el razonamiento anterior de la raíz de la media cuadrática, se puede denotar por $RMS[\Delta\omega]^{(t)} = \sqrt{E[\Delta\omega^2]^{(t)}} + \epsilon$ y así se reemplaza α del denominador, obteniendo la regla de actualizaciones de los parámetros:

$$\begin{aligned}\Delta\omega^{(t)} &= -\frac{RMS[\Delta\omega]^{(t)}}{RMS[g]^{(t)}} \otimes g^{(t)} \\ \omega^{(t+1)} &= \omega^{(t)} - \Delta\omega^{(t)}\end{aligned}$$

Así, desaparece el término *learning rate*.

4.3.7. Resilient backpropagation (Rprop)

Al igual que Adadelta (4.3.6) y Adagrad (4.3.5), Rprop [39] surge para paliar el problema de que los gradientes sean demasiado elevados o demasiado pequeños según la dimensión. Como se usa el aprendizaje por lotes completo, se puede hacer frente a dicho problema utilizando solo el signo del gradiente asegurando así que todas las actualizaciones de peso sean del mismo tamaño lo que ayuda a la optimización en zonas cercanas a puntos de silla y mesetas ya que proporciona variaciones lo suficientemente grandes incluso con pequeñas pendientes.

Rprop combina la idea de utilizar solo el signo del gradiente con la idea de adaptar el tamaño del paso individualmente para cada peso. Sea $\Delta\omega_i^{(t)}$ el tamaño de paso para el i -ésimo parámetro o peso en la t -ésima iteración de la optimización, siendo $\Delta\omega^{(0)}$ y $\Delta\omega^{(1)}$ hiperparámetros del modelo. La regla de actualización de parámetros es:

$$\omega_i^{(t)} = \omega_i^{(t-1)} - \Delta\omega_i^{(t)} \text{sign} \left(g_i^{(t)} \right)$$

donde $\Delta\omega_i^{(t)}$ se calcula según el signo del gradiente de la iteración anterior, en concreto:

$$\Delta\omega_i^{(t)} = \begin{cases} \min \left\{ \lambda \Delta\omega_i^{(t-1)}, \Delta\omega_{max} \right\} & \text{si } g_i^{(t)} g_i^{(t-1)} > 0 \\ \max \left\{ \beta \Delta\omega_i^{(t-1)}, \Delta\omega_{min} \right\} & \text{si } g_i^{(t)} g_i^{(t-1)} < 0 \\ \Delta\omega_i^{(t-1)} & \text{en otro caso} \end{cases}$$

donde λ y $\beta \in \mathbb{R}$ con $\lambda > 1 > \beta$ y $\Delta\omega_{max}, \Delta\omega_{min} \in \mathbb{R}$ son hiperparámetros que evitan los pasos demasiado grandes o demasiado pequeños.

Intuitivamente, cuando el signo del gradiente entre dos iteraciones coincide, el método interpreta que se avanza en la dirección correcta y se debe aumentar el tamaño de la

variación para converger más rápido. Por el contrario, si el signo varía entre dos iteraciones, se ha pasado por encima del óptimo y el tamaño de la variación debe disminuir para evitar volver a saltar sobre dicho óptimo.

Un detalle importante a mencionar es que si para una iteración t se tiene que $g_i^{(t)} g_i^{(t-1)} < 0$, es decir, el signo del gradiente entre dos iteraciones varía, entonces se fija $g_i^{(t)}$ a cero, para evitar una actualización de la tasa de aprendizaje en la siguiente iteración. Los autores lo consideran como un truco de implementación.

4.3.8. RMSprop

El principal problema que presenta Rprop es que no funciona frente a conjuntos de datos muy grandes para los que se necesita realizar actualizaciones de pesos por mini lotes (*mini-batch*).

RMSprop [40] surge motivado por mejorar Rprop, adaptándolo al uso de *mini-batch*. Concretamente hace uso de la misma ecuación que se explicó en la sección 4.3.6, que, manteniendo la notación visto en dicha sección, es la siguiente:

$$\begin{aligned} E[\mathbf{g}^2]^{(t)} &= \gamma E[\mathbf{g}^2]^{(t-1)} + (1 - \gamma) \mathbf{g}^{(t)} \cdot \mathbf{g}^{(t)} \\ \Delta \omega^{(t)} &= -\frac{\alpha}{\sqrt{E[\mathbf{g}^2]^{(t)} + \epsilon}} \otimes \mathbf{g}^{(t)} = -\frac{\alpha}{RMS[\mathbf{g}]^{(t)}} \otimes \mathbf{g}^{(t)} \end{aligned}$$

con $\gamma \in (0, 1)$. Esto es porque el signo del gradiente es igual al gradiente dividido por su norma, y para evitar dividir por un gradiente distinto cada vez al aplicar entrenamiento por *mini-batch*, se aplica un promedio de dichos gradientes.

Así, la diferencia con respecto a Adadelta es que no se realiza ningún cambio en el denominador, es decir, Adadelta autoajusta el hiperparámetro *learning rate* y RMSprop no. Por tanto se entiende como una variante cercana del método Adadelta.

4.3.9. Adam

Adam (*Adaptive Moment Estimation*) [41] es una técnica de optimización que calcula la tasa de aprendizaje para cada uno de los parámetros de forma adaptativa. Almacena los promedios de los gradientes sucesivos y su varianza:

$$\begin{aligned} \mathbf{m}^{(t)} &= \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)} \\ \mathbf{v}^{(t)} &= \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) \mathbf{g}^{(t)} \cdot \mathbf{g}^{(t)} \end{aligned}$$

donde β_1 y β_2 son números reales con $\beta_1, \beta_2 \in (0, 1)$ y de nuevo $\mathbf{g}^{(t)} \cdot \mathbf{g}^{(t)}$ indica la multiplicación elemento a elemento $\mathbf{g}^{(t)} \otimes \mathbf{g}^{(t)}$. Sin embargo, como $\mathbf{m}^{(0)}$ y $\mathbf{v}^{(0)}$ se inicializan

a **0**, las primeras iteraciones quedan sesgadas a cero, por lo que para contrarrestarlo se corrigen de la siguiente forma:

$$\begin{aligned}\hat{\mathbf{m}}^{(t)} &= \frac{\mathbf{m}^{(t)}}{1 - \beta_1^t} \\ \hat{\mathbf{v}}^{(t)} &= \frac{\mathbf{v}^{(t)}}{1 - \beta_2^t}\end{aligned}$$

donde β_1^t y β_2^t denota la potencia t de β_1 y β_2 respectivamente. Así se puede usar ya la regla de actualización de parámetros:

$$\omega^{(t+1)} = \omega^{(t)} - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}^{(t)} + \epsilon}} \otimes \hat{\mathbf{m}}^{(t)}$$

4.3.10. AdaMax

Los propios autores de Adam propusieron en el paper original [41] una versión de dicho método: AdaMax.

En Adam, la regla de actualización para cada peso consiste en escalar sus gradientes de manera inversamente proporcional a una norma ℓ_2 (escalada) de sus gradientes individuales actuales y pasados. Si se generaliza a una norma ℓ_p :

$$\mathbf{v}^{(t)} = \beta_2^p \mathbf{v}^{(t-1)} + (1 - \beta_2^p) |\mathbf{g}^{(t)}|^p = (1 - \beta_2^p) \sum_{i=1}^t \beta_2^{p(t-i)} |\mathbf{g}^{(i)}|^p$$

Las normas para valores p grandes generalmente se vuelven numéricamente inestables. Sin embargo, con la norma ℓ_∞ , que es la que usa AdaMax, se muestra generalmente un comportamiento estable. Así:

$$\begin{aligned}\mathbf{u}^{(t)} &= \lim_{p \rightarrow \infty} (\mathbf{v}^{(t)})^{\frac{1}{p}} = \lim_{p \rightarrow \infty} \left((1 - \beta_2^p) \sum_{i=1}^t \beta_2^{p(t-i)} |\mathbf{g}^{(i)}|^p \right)^{\frac{1}{p}} \\ &= \lim_{p \rightarrow \infty} (1 - \beta_2^p)^{\frac{1}{p}} \left(\sum_{i=1}^t \beta_2^{p(t-i)} |\mathbf{g}^{(i)}|^p \right)^{\frac{1}{p}} \\ &= \lim_{p \rightarrow \infty} \left(\sum_{i=1}^t (\beta_2^{(t-i)} |\mathbf{g}^{(i)}|)^p \right)^{\frac{1}{p}} \\ &= \max \left\{ \beta_2^{(t-1)} |\mathbf{g}^{(1)}|, \beta_2^{(t-2)} |\mathbf{g}^{(2)}|, \dots, \beta_2 |\mathbf{g}^{(t-1)}|, |\mathbf{g}^{(t)}| \right\} \\ &= \max \left\{ \beta_2 \mathbf{u}^{(t-1)}, |\mathbf{g}^{(t)}| \right\}\end{aligned}$$

con $\boldsymbol{u}^{(0)} = \mathbf{0}$. De esta forma la regla de actualización es:

$$\boldsymbol{\omega}^{(t+1)} = \boldsymbol{\omega}^{(t)} - \frac{\alpha}{\boldsymbol{u}^{(t)}} \otimes \hat{\mathbf{m}}^{(t)}$$

4.3.11. Nadam

Nadam (*Nesterov-accelerated Adaptive Moment Estimation*) [42] es una técnica de optimización que combina Adam (4.3.9) y NAG (4.3.4).

Unificando la notación, si $\alpha, \gamma \in \mathbb{R}$ con $\gamma \in (0, 1)$ la regla de actualización de NAG es la siguiente:

$$\begin{aligned}\mathbf{g}^{(t)} &= \nabla[f(\boldsymbol{\omega}^{(t)} - \gamma \mathbf{m}^{(t-1)})] \\ \mathbf{m}^{(t)} &= \gamma \mathbf{m}^{(t-1)} + \alpha \mathbf{g}^{(t)} \\ \boldsymbol{\omega}^{(t+1)} &= \boldsymbol{\omega}^{(t)} - \mathbf{m}^{(t)}\end{aligned}$$

El autor propone una modificación: usar el vector $\mathbf{m}^{(t)}$ directamente para actualizar los parámetros actuales, en vez de usarlo dos veces:

$$\begin{aligned}\mathbf{g}^{(t)} &= \nabla[f(\boldsymbol{\omega}^{(t)})] \\ \mathbf{m}^{(t)} &= \gamma \mathbf{m}^{(t-1)} + \alpha \mathbf{g}^{(t)} \\ \boldsymbol{\omega}^{(t+1)} &= \boldsymbol{\omega}^{(t)} - (\gamma \mathbf{m}^{(t)} - \alpha \mathbf{g}^{(t)})\end{aligned}$$

Por otro lado y de igual manera, se unifica la notación para el método Adam. Sean $\beta_1, \beta_2 \in (0, 1)$:

$$\begin{aligned}\mathbf{m}^{(t)} &= \beta_1 \mathbf{m}^{(t-1)} + (1 - \beta_1) \mathbf{g}^{(t)} \\ \mathbf{v}^{(t)} &= \beta_2 \mathbf{v}^{(t-1)} + (1 - \beta_2) \mathbf{g}^{(t)^2} \\ \hat{\mathbf{m}}^{(t)} &= \frac{\mathbf{m}^{(t)}}{1 - \beta_1^t} \\ \hat{\mathbf{v}}^{(t)} &= \frac{\mathbf{v}^{(t)}}{1 - \beta_2^t} \\ \boldsymbol{\omega}^{(t+1)} &= \boldsymbol{\omega}^{(t)} - \frac{\alpha}{\sqrt{\hat{\mathbf{v}}^{(t)}} + \epsilon} \otimes \hat{\mathbf{m}}^{(t)}\end{aligned}$$

Al expandir la última ecuación con las definiciones de $\hat{\mathbf{m}}^{(t)}$ y $\mathbf{m}^{(t)}$ y simplificando, se obtiene:

$$\begin{aligned}\omega^{(t+1)} &= \omega^{(t)} - \frac{\alpha}{\sqrt{\hat{v}^{(t)} + \epsilon}} \otimes \left(\frac{\beta_1 m^{(t-1)} + (1 - \beta_1) g^{(t)}}{1 - \beta_1^t} \right) \\ &= \omega^{(t)} - \frac{\alpha}{\sqrt{\hat{v}^{(t)} + \epsilon}} \otimes \left(\beta_1 \hat{m}^{(t-1)} + \frac{(1 - \beta_1) g^{(t)}}{1 - \beta_1^t} \right)\end{aligned}$$

Para combinar las dos técnicas simplemente se reemplaza la estimación corregida debido al sesgo de la iteración anterior, $\hat{m}^{(t-1)}$, por la estimación corregida de la iteración actual, $\hat{m}^{(t)}$ (paso análogo al que se realiza al modificar la regla de actualización de NAG, como propone el autor), obteniendo la regla de actualización de parámetros de Nadam:

$$\omega^{(t+1)} = \omega^{(t)} - \frac{\alpha}{\sqrt{\hat{v}^{(t)} + \epsilon}} \otimes \left(\beta_1 \hat{m}^{(t)} + \frac{(1 - \beta_1) g^{(t)}}{1 - \beta_1^t} \right)$$

4.3.12. ¿Qué optimizador usar en cada caso?

Aunque para cada problema la situación varía, analizando las ventajas e inconvenientes de cada optimizados se puede obtener una visión global para saber cuál de ellos puede funcionar mejor en un problema determinado. Para esta visión global se ha consultado el paper [43].

Por un lado, SGD generalmente logra encontrar un mínimo, pero suele llevar mucho más tiempo que con otros optimizadores, es mucho más dependiente de la inicialización y puede quedarse atascado en puntos de silla en lugar de mínimos locales. Además, si existe escasez en los datos de entrada, probablemente se obtengan los mejores resultados utilizando uno de los métodos de tasa de aprendizaje adaptativo.

Por otro lado, RMSprop es una extensión de Adagrad que se ocupa de las tasas de aprendizaje que disminuyen radicalmente. Es idéntico a Adadelta, excepto en que este último usa el RMS de las actualizaciones de parámetros en la regla de actualización del numerador. Adam, finalmente, agrega corrección de sesgos e inercia a RMSprop. Así RMSprop, Adadelta y Adam son algoritmos muy similares que funcionan bien en circunstancias parecidas. El paper original de Adam [41] muestra que la corrección de sesgo ayuda a Adam a superar ligeramente a RMSprop a medida que los gradientes se vuelven más pequeños, por lo que Adam podría ser la mejor opción en general.

Existen una multitud de variantes de optimizadores, sin embargo los estudiados forman el catálogo básico de optimizadores de cualquier *framework* usado para la implementación de redes neuronales, tema en el que se centra el trabajo.

Por otro lado, no se puede dejar sin mencionar que existen otras técnicas de opti-

mización útiles bajo determinadas circunstancias (como que la función de coste no sea derivable), como son las Metaheurísticas, cuyos ejemplos más famosos son los algoritmos genéticos, el enfriamiento simulado, etc.

4.4. Entrenamiento de las redes neuronales *feed-forward*

Una sección que se quedó sin detallar en el capítulo 3 fue estudiar la técnica con la que las redes neuronales son capaces de ajustar los pesos del modelo al desarrollar su aprendizaje, es decir, la técnica de “propagación hacia atrás” (comúnmente conocido por su traducción en inglés: *back-propagation*). En esta sección se profundizará en dicha técnica.

Back-propagation fue la base del artículo *Learning representations by back-propagating errors* [11] publicado 1986 que muestra experimentalmente el uso de dicho algoritmo de aprendizaje para que una red neuronal auto-ajustara sus pesos aprendiendo así una representación interna de la información que se procesa. Desde entonces, junto con sus variantes a distintos tipos de redes neuronales, el algoritmo de *back-propagation* forma una parte indispensable del entrenamiento de cualquier red neuronal.

Situando el problema, si se quiere llevar a cabo el ajuste de los parámetros de una red neuronal se utiliza algún método de optimización (como los estudiados en la sección 4.3) para minimizar la función de coste de la red haciendo uso de la técnica de *back-propagation* para obtener el vector de gradientes dentro de la complejidad de la arquitectura de la red neuronal. Pero, ¿en qué consiste dicho algoritmo?.

Dada una señal de error, en la salida, positiva (si el error es cero no se necesita actualizar los parámetros ya que los actuales son óptimos), la estrategia a seguir es analizar cada neurona y su implicación en el resultado que ha provocado dicha señal de error, penalizando las neuronas que más influencia han tenido en este error. Para hacer dicho análisis de forma eficiente, tiene sentido realizar una retropropagación del error desde la capa de salida hasta las primeras capas, ya que, en una red neuronal, el error de las capas anteriores depende directamente del error de las capas posteriores. Además, es una operación recursiva capa tras capa, transportando el error hacia atrás, lo que implica que, al llegar a la primera capa, con una sola pasada del algoritmo, se conocerá el error de cada neurona y de sus parámetros. Esos errores servirán al algoritmo para calcular las derivadas parciales de cada parámetro de la red con respecto a la función de coste, conformando el vector gradiente, que es lo que necesita el algoritmo de optimización para minimizar el error y por tanto, entrenar a la red. A continuación se va a formalizar dicho algoritmo.

Sea una red neuronal con N entradas $\{x_1, x_2, \dots, x_N\}$, cuya salida se denota con y :

- La j -ésima neurona de la capa k recibe como entrada las salidas de las neuronas de la capa anterior (la capa $k - 1$).
- El parámetro que multiplica a la entrada, proporcionada como salida por la i -ésima neurona de la capa $k - 1$, de la j -ésima neurona de la capa k se denota por $\omega_{i,j}^{(k)}$.
- Para este algoritmo se incluye la separación de la notación del término *bias* de la j -ésima neurona en la capa k ($b_j^{(k)} = \omega_{0,j}^{(k)}$) de los demás parámetros de la suma ponderada ($\omega_{i,j}^{(k)}$ con $i \in \{1, \dots, m\}$, si la capa $k - 1$ está formada por m neuronas).
- Al escalar de salida de la j -ésima neurona de la capa k se denota por $a_j^{(k)}$.
- Por otro lado se denota por $z_j^{(k)}$ al resultado de la suma ponderada de los valores de entrada de la capa k en la j -ésima neurona, es decir, si la capa $k - 1$ está formada por m neuronas entonces $z_j^{(k)} = \omega_{0,j}^{(k)} + \sum_{i=1}^m \omega_{i,j}^{(k)} a_i^{(k-1)}$.
- De esta forma $a_j^{(k)}$ es el resultado de aplicar la función de activación de la j -ésima neurona a la suma ponderada $z_j^{(k)}$.
- Se considera la capa de entrada como la capa 1 en la notación. Así $x_j = z_j^{(1)} = a_j^{(1)} \forall j \in \{1, \dots, N\}$.
- La función de pérdida del modelo se denota como J . El objetivo es calcular la derivada parcial de la función de error con respecto a cualquier parámetro del modelo, es decir:

$$\frac{\partial J}{\omega_{i,j}^{(k)}} \quad \frac{\partial J}{b_j^{(k)}}$$

- Se considera una red donde todas las neuronas de una capa reciben la información de todas las neuronas de la capa anterior.

Ya se puede dar comienzo al algoritmo. Se supone un ejemplo sencillo de red neuronal formada por $L - 2$ capas ocultas y donde cada capa (incluyendo la capa de salida) la compone una neurona. Además solo se tendrá una entrada a la red. Se visualiza un esquema de dicha red en la imagen 4.11. Posteriormente se generalizará dicho algoritmo.

Se supone la función de pérdida J positiva, que cuantifica el error entre la salida de la última capa del modelo y el resultado real observado. Para obtener esta salida de la última capa (denotado por $a_1^{(L)}$) se aplica la operación

$$a_1^{(L)} = \sigma_1^{(L)}(z_1^{(L)}) = \sigma_1^{(L)}(b_1^{(L)} + \omega_{1,1}^{(L)} a_1^{(L-1)}),$$

donde $\sigma_1^{(L)}$ es la función de activación de la primera neurona de la capa L . Por lo tanto,

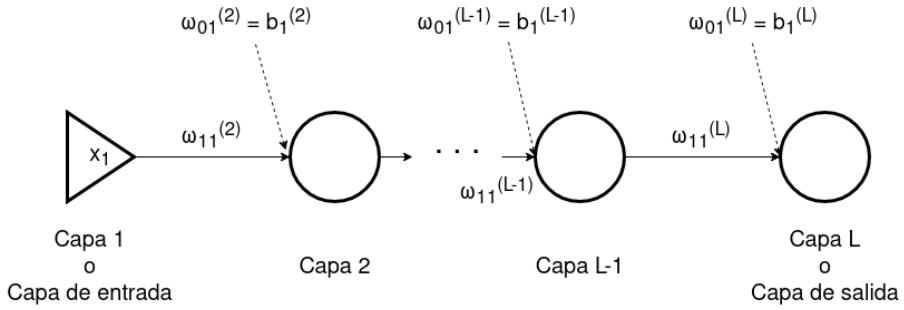


Imagen 4.11: Esquema de la red neuronal usada para el ejemplo. Está formada por \$L - 2\$ capas ocultas y donde cada capa (incluyendo la capa de salida) la compone una neurona. Además solo se tendrá una entrada a la red.

se puede aplicar la regla de la cadena a la composición \$J \circ a_1^{(L)} \circ z_1^{(L)}\$:

$$\frac{\partial J}{\partial b_1^{(L)}} = \underbrace{\frac{\partial J}{\partial a_1^{(L)}} \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}}}_{\delta_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial b_1^{(L)}} \quad \frac{\partial J}{\partial \omega_{1,1}^{(L)}} = \underbrace{\frac{\partial J}{\partial a_1^{(L)}} \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}}}_{\delta_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial \omega_{1,1}^{(L)}} \quad (4.3)$$

Como se puede observar hay dos términos que comparten las dos derivadas, como era esperable. Ese valor representa cómo varía el error en función de la suma ponderada de los parámetros calculada dentro de las neuronas, es decir, en qué grado se modifica el error cuando se produce un pequeño cambio en la suma de la neurona. En resumen, esa derivada devuelve el error imputado a la neurona y se denota con el símbolo \$\delta_i^{(k)}\$ para la \$i\$-ésima neurona de la capa \$k\$ (en este ejemplo \$\delta_1^{(L)}\$). Este término dependerá de la definición de la función de coste escogida y de la función de activación de la neurona correspondiente.

Por otro lado como \$\frac{\partial z_1^{(L)}}{\partial b_1^{(L)}} = 1\$ y \$\frac{\partial z_1^{(L)}}{\partial \omega_{1,1}^{(L)}} = a_1^{(L-1)}\$, se pueden simplificar las expresiones anteriores (4.3) como:

$$\frac{\partial J}{\partial b_1^{(L)}} = \delta_1^{(L)} \quad \frac{\partial J}{\partial \omega_{1,1}^{(L)}} = \delta_1^{(L)} a_1^{(L-1)}$$

Una vez logrado el objetivo para la última capa (capa \$L\$), se procederá a calcular las derivadas del error con respecto a los parámetros de la capa anterior (capa \$L-1\$). Siguiendo el mismo procedimiento, la salida de la red en la penúltima capa es el resultado

de la composición $J \circ a_1^{(L)} \circ z_1^{(L)} \circ a_1^{(L-1)} \circ z_1^{(L-1)}$:

$$\frac{\partial J}{\partial b_1^{(L-1)}} = \overbrace{\underbrace{\frac{\partial J}{\partial a_1^{(L)}} \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial a_1^{(L-1)}} \frac{\partial a_1^{(L-1)}}{\partial z_1^{(L-1)}} \frac{\partial z_1^{(L-1)}}{\partial b_1^{(L-1)}}}_{\delta_1^{(L)}}}^{\delta_1^{(L-1)}}$$

$$\frac{\partial J}{\partial \omega_{1,1}^{(L-1)}} = \overbrace{\underbrace{\frac{\partial J}{\partial a_1^{(L)}} \frac{\partial a_1^{(L)}}{\partial z_1^{(L)}} \frac{\partial z_1^{(L)}}{\partial a_1^{(L-1)}} \frac{\partial a_1^{(L-1)}}{\partial z_1^{(L-1)}} \frac{\partial z_1^{(L-1)}}{\partial \omega_{1,1}^{(L-1)}}}_{\delta_1^{(L)}}}^{\delta_1^{(L-1)}}$$
(4.4)

Analizando la ecuación anterior, se ve de nuevo que existen dos partes coincidentes en ambas derivadas: al conjunto de ellas se denota por $\delta_1^{(L-1)}$ (al referirse a la capa $L-1$) que vuelve a representar el error de la neurona en dicha capa. Pero además esto es muy sencillo de calcular: por un lado aparece el error imputado a la neurona de la capa anterior ($\delta_1^{(L)}$), que ya está calculado; por otro lado, $\frac{\partial a_1^{(L-1)}}{\partial z_1^{(L-1)}}$ es la derivada de la función de activación de la neurona de la capa $K-1$ con respecto a la suma ponderada. Por tanto solo falta por conocer la derivada $\frac{\partial z_1^{(L)}}{\partial a_1^{(L-1)}}$ que indica como varía la suma ponderada de una capa cuando se modifica la salida de la capa anterior, que es fácil de calcular (como $z_1^{(L)} = b_1^{(L)} + \omega_{1,1}^{(L)} a_1^{(L-1)}$, entonces $\frac{\partial z_1^{(L)}}{\partial a_1^{(L-1)}} = \omega_{1,1}^{(L)}$). Y por último, de nuevo, $\frac{\partial z_1^{(L-1)}}{\partial \omega_{1,1}^{(L-1)}} = a_1^{(L-2)}$ y $\frac{\partial z_1^{(L-1)}}{\partial b_1^{(L-1)}} = 1$. Así simplificando en 4.4:

$$\frac{\partial J}{\partial b_1^{(L-1)}} = \delta_1^{(L-1)} \quad \frac{\partial J}{\partial \omega_{1,1}^{(L-1)}} = \delta_1^{(L-1)} a_1^{(L-2)}$$

donde $\delta_1^{(L-1)} = \omega_{1,1}^{(L)} \delta_1^{(L)} \frac{\partial a_1^{(L-1)}}{\partial z_1^{(L-1)}}$.

Lo anteriormente discutido para esta última capa ya se puede extender al resto de las capas de la red, aplicando la misma lógica: sea una capa k de la red neuronal, se toma el error de la capa posterior $\delta_1^{(k+1)}$ y se retropropaga a la capa actual,

$$\delta_1^{(k)} = \omega_{1,1}^{(k+1)} \delta_1^{(k+1)} \frac{\partial a_1^{(k)}}{\partial z_1^{(k)}}$$

para, con esto, poder calcular las derivadas parciales de la función de pérdida con respecto a cada parámetro de la capa:

$$\frac{\partial J}{\partial b_1^{(k)}} = \delta_1^{(k)} \quad \frac{\partial J}{\partial \omega_{1,1}^{(k)}} = \delta_1^{(k)} a_1^{(k-1)}$$

Así, haciendo solo uso de cuatro expresiones se pueden calcular las derivadas parciales de todos los parámetros de la red neuronal en un solo pase hacia atrás.

Se puede generalizar dicho procedimiento a una red de múltiples neuronas por capa, simplemente modificando algunos índices. Se recuerda que el resultado devuelto por la j -ésima neurona de la capa k viene dado por:

$$a_j^{(k)} = \sigma_j^{(k)} (z_j^{(k)}) = \sigma_j^{(k)} \left(b_j^{(k)} + \sum_{i=1}^{m_{k-1}} \omega_{i,j}^{(k)} a_i^{(L-1)} \right),$$

donde m_{k-1} es la cantidad de neuronas que forma la capa $k - 1$. Así, aplicando la regla de la cadena a las operaciones de la última capa (L):

$$\frac{\partial J}{\partial b_j^{(L)}} = \underbrace{\frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial b_j^{(L)}}}_{\delta_j^{(L)}} \quad \frac{\partial J}{\partial \omega_{i,j}^{(L)}} = \underbrace{\frac{\partial J}{\partial a_j^{(L)}} \frac{\partial a_j^{(L)}}{\partial z_j^{(L)}} \frac{\partial z_j^{(L)}}{\partial \omega_{i,j}^{(L)}}}_{\delta_j^{(L)}} \quad (4.5)$$

Y como $\frac{\partial z_j^{(L)}}{\partial b_j^{(L)}} = 1$ y $\frac{\partial z_j^{(L)}}{\partial \omega_{i,j}^{(L)}} = a_i^{(L-1)}$, se simplifican, de nuevo, las expresiones anteriores (4.5) como:

$$\frac{\partial J}{\partial b_j^{(L)}} = \delta_j^{(L)} \quad \frac{\partial J}{\partial \omega_{i,j}^{(L)}} = \delta_j^{(L)} a_i^{(L-1)}$$

La diferencia notable con respecto al ejemplo sencillo 4.11 es que el error entre capas se propaga por diferentes caminos, así:

$$\frac{\partial J}{\partial a_j^{(L-1)}} = \sum_{i=1}^{m_L} \frac{\partial J}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial z_i^{(L)}} \frac{\partial z_i^{(L)}}{\partial a_j^{(L-1)}}$$

Usando dicha igualdad, se tiene:

$$\frac{\partial J}{\partial b_j^{(L-1)}} = \underbrace{\frac{\partial J}{\partial a_j^{(L-1)}} \frac{\partial a_j^{(L-1)}}{\partial z_j^{(L-1)}} \frac{\partial z_j^{(L-1)}}{\partial b_j^{(L-1)}}}_{\delta_j^{(L-1)}}$$

$$\frac{\partial J}{\partial \omega_{i,j}^{(L-1)}} = \underbrace{\frac{\partial J}{\partial a_j^{(L-1)}} \frac{\partial a_j^{(L-1)}}{\partial z_j^{(L-1)}} \frac{\partial z_j^{(L-1)}}{\partial \omega_{i,j}^{(L-1)}}}_{\delta_j^{(L-1)}}$$

De nuevo, se simplifican dichas expresiones usando que $\frac{\partial z_j^{(L-1)}}{\partial b_j^{(L-1)}} = 1$ y $\frac{\partial z_j^{(L-1)}}{\partial \omega_{i,j}^{(L-1)}} = a_i^{(L-2)}$:

$$\frac{\partial J}{\partial b_j^{(L-1)}} = \delta_j^{(L-1)} \quad \frac{\partial J}{\partial \omega_{i,j}^{(L-1)}} = \delta_j^{(L-1)} a_i^{(L-2)}$$

Y así respectivamente con todas las capas obteniendo, para una capa $k < L$:

$$\frac{\partial J}{\partial a_j^{(k)}} = \sum_{i=1}^{m_{k+1}} \frac{\partial J}{\partial a_i^{(k+1)}} \frac{\partial a_i^{(k+1)}}{\partial z_i^{(k+1)}} \frac{\partial z_i^{(k+1)}}{\partial a_j^{(k)}}$$

Usando dicha igualdad, se tiene:

$$\begin{aligned} \frac{\partial J}{\partial b_j^{(k)}} &= \underbrace{\frac{\partial J}{\partial a_j^{(k)}} \frac{\partial a_j^{(k)}}{\partial z_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial b_j^{(k)}}}_{\delta_j^{(k)}} \\ \frac{\partial J}{\partial \omega_{i,j}^{(k)}} &= \underbrace{\frac{\partial J}{\partial a_j^{(k)}} \frac{\partial a_j^{(k)}}{\partial z_j^{(k)}} \frac{\partial z_j^{(k)}}{\partial \omega_{i,j}^{(k)}}}_{\delta_j^{(k)}} \end{aligned} \tag{4.6}$$

Y usando por última vez que $\frac{\partial z_j^{(k+1)}}{\partial b_j^{(k+1)}} = 1$ y $\frac{\partial z_j^{(k+1)}}{\partial \omega_{i,j}^{(k)}} = a_i^{(k)}$, se simplifican, de nuevo, las expresiones anteriores (4.6) como:

$$\frac{\partial J}{\partial b_j^{(k)}} = \delta_j^{(k)} \quad \frac{\partial J}{\partial \omega_{i,j}^{(k)}} = \delta_j^{(k)} a_i^{(k-1)} \tag{4.7}$$

Así, haciendo solo uso de estas últimas expresiones se pueden calcular las derivadas parciales de todos los parámetros de la red neuronal en un solo pase hacia atrás. Estas derivadas componen el vector gradiente que es lo que necesita el método de optimización (véase la sección 4.3) para llevar a cabo su tarea y autoajustar los parámetros del modelo.

En la práctica, los diferentes *frameworks* y bibliotecas (como Theano, Autograd, TensorFlow y PyTorch) permiten automatizar dicho procedimiento de forma sencilla mediante la diferenciación automática, un procedimiento de cálculo automático de las derivadas en un punto.

Dicho procedimiento se basa en que una función puede verse como una composición de operaciones elementales para las cuales se conoce su derivada. Estas operaciones generan lo que se denomina un grafo computacional. Así el algoritmo recorre el grafo de la función y guarda los valores de cada variable y las dependencias entre los nodos. Más tarde se recorre de nuevo el grafo pero empezando por el final y las derivadas de cada variable se calculan respecto a la variable que le antecede.

4.5. Entrenamiento de las redes neuronales recurrentes

De forma análoga al entrenamiento de redes neuronales *feed-forward*, se utiliza, de nuevo, un método de optimización para minimizar la función de coste de la red haciendo uso, en este caso, de una variante de la técnica de *back-propagation* para obtener el vector de gradientes [15, 16].

Lo primero que hay que mencionar es que la función de coste de la red J se basa en una función de coste en cada instante de tiempo. Por ejemplo, si la función de pérdida usada fuese la entropía cruzada, sean \mathcal{V} el vocabulario de estudio, $\hat{\mathbf{y}}^{(t)}$ la palabra que el modelo ha predicho en el instante t (usando representación *One-Hot Encoding*) e $\mathbf{y}^{(t)}$ la verdadera palabra en el instante t (con la misma representación):

$$J^{(t)}(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = - \sum_{\omega \in \mathcal{V}} y_{\omega}^{(t)} \log \hat{y}_{\omega}^{(t)}$$

Entonces la función de pérdida total del modelo, suponiendo T instantes de tiempo, sería:

$$J(\mathbf{y}, \hat{\mathbf{y}}) = \sum_{t=1}^T J^{(t)}(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)})$$

donde \mathbf{y} es la secuencia de palabras esperada y $\hat{\mathbf{y}}$ la predicha por el modelo.

Por otro lado, la variante de *back-propagation* usada para el entrenamiento de redes neuronales recurrentes básicas se denomina “propagación hacia atrás a través del tiempo” (*back-propagation through time*, en inglés).

Manteniendo la notación vista en la sección 3.5.2, para cada instante de tiempo $t \in \{1, \dots, T\}$, una red neuronal aplica las siguientes ecuaciones:

$$\begin{aligned}
z^{(t)} &= b_1 + W_h h^{(t-1)} + W_x x^{(t)} \\
h^{(t)} &= \sigma_1(z^{(t)}) \\
o^{(t)} &= b_2 + U h^{(t)} \\
\hat{y}^{(t)} &= \sigma_2(o^{(t)})
\end{aligned}$$

Así, aplicando la regla de la cadena y reglas de derivación, se tiene que:

$$\begin{aligned}
\frac{\partial J^{(t)}}{\partial U} &= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial U} \\
\frac{\partial J^{(t)}}{\partial b_2} &= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial b_2}
\end{aligned}$$

Por otro lado, el problema que se presenta es el de calcular W_x , b_1 y W_h ya que existe una dependencia recursiva del estado anterior. Como $h^{(t)}$ está en función de $h^{(t-1)}$ que contiene a W_x , la regla de la cadena dice que:

$$\begin{aligned}
\frac{\partial J^{(t)}}{\partial W_x} &= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial W_x} + \\
&+ \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial W_x} + \\
&+ \dots + \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial h^{(0)}} \frac{\partial h^{(0)}}{\partial W_x} = \\
&= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial W_x} + \\
&+ \sum_{i=0}^{t-1} \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial h^{(i)}} \frac{\partial h^{(i)}}{\partial W_x}
\end{aligned}$$

De forma análoga obtenemos que:

$$\begin{aligned}
\frac{\partial J^{(t)}}{\partial b_1} &= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial b_1} + \\
&+ \sum_{i=0}^{t-1} \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial z^{(t)}} \frac{\partial z^{(t)}}{\partial h^{(i)}} \frac{\partial h^{(i)}}{\partial b_1}
\end{aligned}$$

y que:

$$\begin{aligned}\frac{\partial J^{(t)}}{\partial W_h} &= \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial \mathbf{o}^{(t)}} \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial \mathbf{z}^{(t)}} \frac{\partial \mathbf{z}^{(t)}}{\partial W_h} + \\ &+ \sum_{i=0}^{t-1} \frac{\partial J^{(t)}}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial \mathbf{o}^{(t)}} \frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial \mathbf{z}^{(t)}} \frac{\partial \mathbf{z}^{(t)}}{\partial \mathbf{h}^{(i)}} \frac{\partial \mathbf{h}^{(i)}}{\partial W_h}\end{aligned}$$

Para calcular el error total simplemente habría que sumar todos los errores en cada instante de tiempo.

De nuevo, en la práctica, se hará uso de la diferenciación automática para obtener los resultados de dichas derivadas.

4.6. Programación diferenciable

En el campo del *deep learning* está eclosionando un enfoque distinto de programación a la hora de diseñar una arquitectura de red neuronal: apilar distintos bloques funcionales parametrizados, permitiéndoles realizar modificaciones de forma dinámica en función de los datos de entrada. Además estos bloques diferenciables introducen capacidades como atención, memoria, etc. y permiten que el modelo se ajuste a cada situación y tarea concreta. Dicho enfoque es lo que se denomina programación diferenciable (*differentiable programming*, en inglés) [53].

Los mecanismos de atención (vistos en la sección 3.5.4) y las redes de memoria (como las vistas en la sección 3.5.3) son ejemplos de este nuevo paradigma de programación.

Este enfoque ha eclosionado gracias a la evolución y creación de bibliotecas y *frameworks* de software que realizan de forma sencilla técnicas como la diferenciación automática, que permiten usar las capacidades computacionales de las Unidades de Procesamiento de Gráficos (GPUs) y facilitan la creación de bloques y arquitecturas neuronales de manera eficiente.

El cambio de perspectiva se puede ilustrar de la siguiente forma: mientras que en el *deep learning* se usan capas de redes neuronales de manera tradicional, en la programación diferenciable se amplían las capacidades de percepción usando cualquier componente diferenciable que aporte atención, razonamiento, etc. A fin de cuentas, la programación diferenciable es una generalización práctica del *deep learning* que, además, eleva su nivel de abstracción.

4.7. Teorema de aproximación universal

Por último, para completar la base teórica matemática de las redes neuronales y comprender la popularidad que han obtenido en éstos años, en esta última sección del capítulo,

se va a demostrar la idoneidad de la arquitectura neuronal para la aproximación de funciones. Para esto se hará uso del teorema de aproximación universal. Dicho teorema posee dos enfoques: el caso de ancho arbitrario, que caracteriza la capacidad de una red neuronal de aproximar funciones con un número arbitrario de neuronas, que, para estudiarlo, se seguirá el desarrollo que realizó Cybenko [49]; y el caso de profundidad arbitraria, que estudia la capacidad de una red neuronal de aproximar funciones con un número arbitrario de capas donde cada una de ellas tiene un número limitado de neuronas, para lo que se usarán dos estudios recientes [51, 52].

4.7.1 Definición. Sea I_n el cubo unidad de dimensión n ($I_n = [0, 1]^n$) y $M(I_n)$ el espacio de las medidas de Borel con signo, finitas y regulares en I_n . Se dice que una función $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ es *discriminatoria* si para una medida $\mu \in M(I_n)$ que cumpla que

$$\int_{I_n} \sigma(\omega^\top x + b) d\mu(x) = 0, \forall x \in \mathbb{R}^n \text{ y } \forall b \in \mathbb{R},$$

entonces $\mu = 0$.

4.7.2 Teorema (Teorema 1 de Cybenko). Si $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ es una función continua y discriminatoria, $N \in \mathbb{N}$, α_j y $b_j \in \mathbb{R}$ y ω_j y $x \in \mathbb{R}^n \forall j \in \{1, \dots, N\}$, entonces el conjunto de sumas finitas $G(x) = \sum_{j=1}^N \alpha_j \sigma(\omega_j^\top x + b_j)$ es denso en $C(I_n)$, es decir, es denso en el espacio de funciones continuas en I_n .

Demostración

Se puede consultar en [49].

4.7.3 Definición. Se dice que una función $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ es *sigmoidal* si:

$$\lim_{x \rightarrow a} \sigma(x) = \begin{cases} 0 & \text{si } a = \infty \\ 1 & \text{si } a = -\infty \end{cases}$$

4.7.4 Lema (Lema 1 de Cybenko). Sea σ una función sigmoidal, medible y acotada, entonces es discriminatoria. En particular cualquier función sigmoidal continua es discriminatoria.

Demostración

Se puede consultar en [49].

Usando la combinación del Teorema 4.7.2 junto con el Lema 4.7.4, Cybenko muestra, en el siguiente teorema, que las redes con una capa oculta y una función sigmoidal continua como función de activación pueden aproximar funciones continuas siempre que no se

impongan restricciones en el número de neuronas o el valor de los pesos.

4.7.5 Teorema (Teorema de aproximación universal, ancho arbitrario (versión 1)). *Si $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ es una función continua sigmoidal, $N \in \mathbb{N}$, α_j y $b_j \in \mathbb{R}$ y ω_j y $\mathbf{x} \in \mathbb{R}^n \forall j \in \{1, \dots, N\}$, entonces el conjunto de sumas finitas $G(\mathbf{x}) = \sum_{j=1}^N \alpha_j \sigma(\omega_j^\top \mathbf{x} + b_j)$ es denso en $C(I_n)$.*

Demostración

Combinación directa de 4.7.4 con 4.7.2.

El siguiente resultado, demostrado en 1993 [50], generaliza el anterior rebajando la restricción de que la función σ sea sigmoidal a, sólo, que no sea polinómica. Para llegar a él se necesitan unas definiciones previas:

4.7.6 Definición. *Fijado un conjunto abierto $\Omega \neq \emptyset$ de \mathbb{R}^n , σ una función medible de Ω en un cuerpo \mathbb{K} , se dice que σ es una función esencialmente acotada en Ω cuando existe $M \in \mathbb{R}_0^+$ tal que $|\sigma(\mathbf{x})| \leq M$ para casi todo $\mathbf{x} \in \Omega$. Al conjunto de todas las funciones esencialmente acotadas en Ω se le denota como $L_\infty(\Omega)$.*

4.7.7 Definición. *Fijado un conjunto abierto $\Omega \neq \emptyset$ de \mathbb{R}^n , σ una función medible de Ω en un cuerpo \mathbb{K} , se dice que σ es una función localmente esencialmente acotada en Ω cuando para cada conjunto compacto $D \subset \Omega$ se tiene que $\sigma \in L_\infty(D)$.*

4.7.8 Teorema (Teorema de aproximación universal, ancho arbitrario (versión 2)). *Sea M el conjunto de funciones localmente esencialmente acotadas en \mathbb{R} que cumplen que el cierre del conjunto de puntos de discontinuidad de dichas funciones tienen medida nula, $N \in \mathbb{N}$, α_j y $b_j \in \mathbb{R}$ y ω_j y $\mathbf{x} \in \mathbb{R}^n \forall j \in \{1, \dots, N\}$, entonces para cualquier función $\sigma \in M$, el conjunto de sumas finitas $G(\mathbf{x}) = \sum_{j=1}^N \alpha_j \sigma(\omega_j^\top \mathbf{x} + b_j)$ es denso en $C(I_n)$.*

Demostración

Se puede consultar en [50].

Con este desarrollo se prueba que una red neuronal multicapa estándar con una función de activación continua por partes localmente acotada puede aproximar cualquier función continua con cualquier grado de precisión si y solo si la función de activación de la red no es polinómica atendiendo al número de neuronas.

Por otro lado el caso de profundidad arbitraria es resultado de estudios mas recientes. La versión probada en [51] demuestra que las redes de ancho $n+4$ con funciones de activación ReLU pueden逼近arse a cualquier función integrable de Lebesgue en el espacio de entrada de dimensión n con respecto a la distancia L_1 si no se ponen restricciones

a la profundidad de la red.

4.7.9 Teorema (Teorema de aproximación universal, profundidad arbitraria (versión 1)). *Para cualquier función integrable de Lebesgue $\sigma : \mathbb{R}^n \rightarrow \mathbb{R}$ y para cualquier $\epsilon > 0$ existe una red neuronal totalmente conectada cuyas neuronas poseen la función de activación ReLU, con ancho $d_m \leq n - 4$ de forma que la función F representada por la red satisface que:*

$$\int_{\mathbb{R}^n} |\sigma(\mathbf{x}) - F(\mathbf{x})| d\mathbf{x} < \epsilon$$

Demostración

Se puede consultar en [51].

Desarrollos más recientes rebajan aún más las restricciones de elección de la función de activación, permitiendo incluso usar polinomios. El siguiente desarrollo se incluye en [52]:

4.7.10 Teorema (Teorema de aproximación universal, profundidad arbitraria (versión 2)). *Sea $\sigma : \mathbb{R} \rightarrow \mathbb{R}$ cualquier función no afín continua que es continuamente diferenciable en al menos un punto en el que la derivada en dicho punto no sea cero, $K \subseteq \mathbb{R}^n$ un conjunto compacto, entonces la clase de funciones definidas de $\mathbb{R}^n \rightarrow \mathbb{R}^m$ que son capaces de ser representadas por redes neuronales con n neuronas en la capa de entrada, m neuronas en la capa de salida y un número arbitrario de capas ocultas, cada una de ellas con $n+m+2$ neuronas con función de activación σ y las neuronas de capa de salida con la identidad como función de activación, es densa en $C(K; \mathbb{R}^m)$ con respecto a la norma del supremo.*

Demostración

Se puede consultar en [52].

En resumen, el teorema de aproximación universal, con sus distintas variantes, demuestra que las redes neuronales pueden aproximar un gran catálogo de funciones. Sin embargo, no proporcionan un método efectivo de creación de la arquitectura de la red para que la aproximación sea posible, sino que simplemente demuestran que tal arquitectura efectiva existe.

Capítulo 5

Experimentación

En este capítulo se desarrollará y analizará una aplicación práctica conjunta de los dos capítulos anteriores (3 y 4). En concreto se procederá a la generación de diversos textos, examinando sus características.

5.1. Propósito y breve introducción

El propósito de esta experimentación será la de analizar las capacidades de varios de los modelos del lenguaje anteriormente estudiados (véase la sección 3.5.5) frente a varias tareas del *Natural Language Processing*.

En particular, se llevarán a cabo dos experimentos:

Experimento 1: Se aplicará el modelo GPT-2 para la tarea de generación de discursos que simulen ser narrados por la reina de Inglaterra. Para ello se extraerá un corpus de discursos atribuidos a este personaje, se aplicará la técnica de *fine-tuning* al modelo GPT-2 con este corpus y se generan diferentes pruebas.

Experimento 2: Se aplicará el modelo BERT para la tarea de completar oraciones de forma coherente. En este caso, el idioma del modelo y de la tarea será el español, ofreciendo así variedad y permitiendo la comparativa entre la realización de tareas con diferentes idiomas.

5.2. Herramientas usadas

La experimentación se ha desarrollado en Python haciendo uso del entorno de Google Colab, un entorno gratuito de Jupyter Notebook (a su vez, otro entorno interactivo que se ejecuta en la web como una aplicación cliente-servidor) desarrollado por Google. Esta plataforma resulta de gran utilidad pues permite ejecutar código en su nube ofreciendo,

además, la posibilidad de hacer uso de sus GPUs.

Entrando en detalles, Colab brinda un servicio gratuito de la GPU Nvidia Tesla K80. El servicio en la nube permite usar dicha GPU para entrenar los modelos neuronales de forma más rápida que en el ordenador del usuario.

Por otro lado, la base de la experimentación realizada tiene origen en la biblioteca [HuggingFace Transformers](#) [55] que ofrece arquitecturas y marcos de trabajo previamente entrenados para el *Natural Language Processing*. Destaca la gran cantidad de modelos previamente entrenados en una gran cantidad de idiomas. Proporciona una API para descargar y usar de forma inmediata dichos modelos y módulos en Python que se pueden modificar para realizar los experimentos oportunos.

La biblioteca está cimentada en dos de las bibliotecas de *deep learning* más populares: PyTorch [56] y TensorFlow [57]. En concreto, el código que se usará en esta experimentación estará basado en PyTorch.

5.3. Experimento 1

5.3.1. Diseño experimental

El primer experimento realizado consiste en generar un discurso de la reina de Inglaterra, Isabel II. Para su ejecución se seguirá el esquema dada en la figura 5.1. Por un lado se obtendrá el corpus formado por 301 discursos de la casa real británica. A continuación de realizará una manipulación de dichos datos para que se ajusten al modelo elegido. Además se aplicará la técnica de *fine-tuning* al modelo con dicho corpus. Por último, se generarán varios discursos y se analizarán las características comunes de los textos generados y las específicas de alguno en particular.

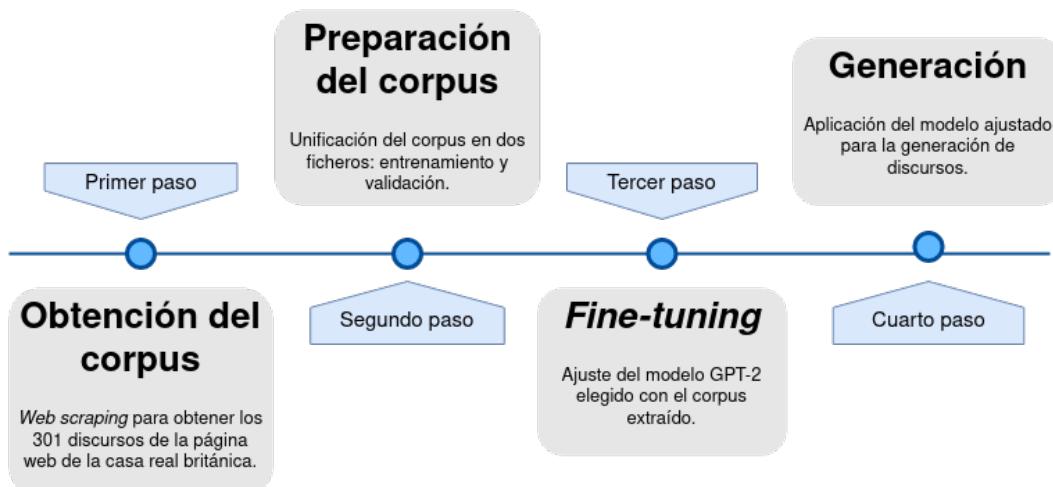


Imagen 5.1: Esquema del diseño del experimento 1.

Obtención y preparación del corpus

Para este experimento se han utilizado los 301 discursos de la reina de Inglaterra ofrecidos por la [página web de la casa real británica](#) (a fecha de 22 de octubre de 2020) [54].

Para extraerlos se utilizó la técnica de *web scraping* (que consiste en navegar automáticamente por una web y extraer información de ella). No se detallará aquí el procedimiento seguido, ya que no es objeto de estudio del proyecto, pero todos los detalles están disponibles en el [código fuente](#).

Una vez extraídos los discursos, se unifica el corpus en dos únicos ficheros: uno para el entrenamiento del modelo y otro para la validación de este. Se ha considerado un valor típico: 70 % de los discursos para entrenamiento y 30 % de los discursos para validación. Hay que tener en cuenta que el corpus total cuenta con un número aproximado de 140000 palabras de las cuales cerca de 100000 se han dedicado a entrenamiento y 40000 a validación, componiendo una cantidad cercana a 1 MB de datos. Es un corpus muy pequeño en comparación con los datos de entrenamiento que se usan para entrenar los modelos usados (por ejemplo, GPT-2 usó 45 millones de páginas web con un tamaño total de 40 GB entrenamiento), por lo que se partirá de un modelo previamente entrenado y se le aplicará la técnica de *fine-tuning* (explicada en la sección 3.5.6) con dicho corpus.

Fine-tuning de los modelos usados

Se ajustará, al conjunto de datos expuesto, uno de los modelos introducidos en la sección 3.5.5, concretamente [GPT-2](#). La versión usada es la versión simplificada formada por 12 capas con 768 estados ocultos y 117 millones de parámetros, que está disponible, previamente entrenada con textos en inglés, en la biblioteca [Transformers](#).

Para este proceso de *fine-tuning* se partirá del script que proporciona la biblioteca *Transformers* para dicha tarea: [run_language_modeling.py](#) [55]. Este programa realiza el *fine-tuning* de los modelos previamente entrenados que se le indique sobre el corpus de texto que se le pase como argumento.

Dicho script recibe varios argumentos por la línea de comandos relacionados con qué modelo se va a ajustar (*model_name_or_path*, *model_type*, y otros que no se usarán), relacionados con los datos que se van a introducir en el modelo para entrenamiento y evaluación (*train_data_file*, *eval_data_file*, y otros que no se usarán) y relacionados con la configuración del entrenamiento del modelo y con la obtención de resultados (*output_dir*, *overwrite_output_dir*, *do_train*, *do_eval*, *per_device_train_batch_size*, *per_device_eval_batch_size*, *evaluate_during_training*, *learning_rate*, *num_train_epochs*, entre otros que no se usarán). Para conocer una explicación detallada de cada uno de los argumentos se puede ejecutar el

script `run_language_modeling.py` con el argumento `help`.

Se usará el optimizador Adam (explicado en la sección 4.3.9) para minimizar el error provocado por las tareas de modelado de lenguaje causal (*causal language modeling*, CLM). La tarea de modelado de lenguaje causal consiste en modelar la probabilidad de una palabra dadas las palabras anteriores en una oración.

De esta forma, una vez ejecutado el script con los parámetros necesarios para cada configuración del modelo (visite el [código](#) para visualizar cómo se produce esta llamada al script con los distintos argumentos necesarios) se obtendrán los modelos ajustados preparados para generar texto.

Generación de texto

Para la tarea de generación de texto se usará otro script que proporciona la biblioteca *Transformers* destinada a realizar dicho ejercicio: [`run_generation.py`](#) [55].

Se ha adaptado dicho script para que permita la posibilidad de guiar las ideas del discurso generado mediante varias semillas (tantas como se quiera). Para ello simplemente hay que incluir las veces que se desee el argumento `prompt` indicándole la semilla que se desee. El script modificado se puede consultar en el fichero [`run_generation.py`](#).

El script modificado para la generación de texto, igual que el de *fine-tuning*, recibe distintos argumentos por línea de comandos, entre los que destacan: `model_name_or_path` y `model_type` para indicarle el modelo que se va a usar para la generación de texto (en este caso GPT-2, con la consiguiente ruta al directorio donde se encuentra el ajuste de *fine-tuning* realizado anteriormente); `prompt`, que indica las semillas (en forma de texto) desde donde parte el generador para completar el discurso; y otras opciones de generación como `temperature`, `repetition_penalty`, `k`, `seed`, `length` o `stop_token`. Se pueden consultar los posibles argumentos con su explicación detallada ejecutando el script con el argumento `help`.

De esta manera, si se ejecuta el script con los parámetros y semillas deseados se obtendrá un discurso generado mediante Inteligencia Artificial y puesto en boca de la reina de Inglaterra. Puede visitar el [código](#) para visualizar cómo se producen algunos ejemplos de discursos y las respectivas llamadas al script de generación de texto con los argumentos necesarios.

5.3.2. Resultados

A continuación se van a incluir algunos extractos de los discursos generados más curiosos y característicos. Los textos completos (de mayor calidad) se incluyen en el anexo A del proyecto.

Primera inicialización de las semillas

Sin fine-tuning

Para obtener dichos textos se le ha introducido tres semillas: “*The pandemic generated by the COVID-19 virus has engulfed our nation since February of this year*”, “*Furthermore, the withdrawal of the United Kingdom from the European Union*” y “*We are immersed in a period where our nation must get the best of itself and fight*”.

Para conocer la situación de partida se ha ejecutado el generador partiendo del modelo GPT-2 original, sin haber realizado *fine-tuning* con el corpus de discursos. De entre todas las pruebas realizadas se han escogido dos discursos que han resultado más interesantes para su análisis: A.1.1 y A.1.2

The pandemic generated by the COVID-19 virus has engulfed our nation since February of this year, which triggered an enormous surge in emergency preparedness, health care, disaster relief and medical treatment. It is now happening again in Afghanistan and elsewhere.

The United States has put billions in emergency health funding in Afghanistan, and is now able to mobilize the United States Government and the international community to fight this epidemic. ...

(Extracto del discurso A.1.1)

Los discursos parten de la primera semilla introducida como argumento a la llamada del script. A partir de esta, se pretende seguir el hilo argumental relacionado a dicha semilla.

En este caso, al introducir un tema relacionado con un virus y una pandemia, el modelo ha sabido relacionarlo con otras entidades del campo sanitario como “preparación para emergencias” (*emergency preparedness*), “atención médica” (*health care o medical treatment*) y que la semilla introducida provoca la situación de un desastre (*disaster*) o epidemia (*epidemic*).

Por otro lado, también introduce entidades en forma de naciones o países como Estados Unidos y Afganistán (*The United States* y *Afghanistan*). Esto puede ser debido a introducir en la semilla la secuencia de palabras “nuestra nación” (*our nation*).

Como se puede apreciar, el modelo no tiene problema en reconocer y relacionar entidades, permitiendo un uso correcto de ellas y generando un texto coherente a partir de ellas.

... President Obama has the resources and will to provide these assistance to our nation, our partners, and all the world's people. ...

(Extracto del discurso A.1.1)

Para entender frases como esta hay que recordar que el modelo original ha sido entrenado antes de febrero de 2019 con texto que ya existía en las diferentes páginas web donde fue entrenado. Por eso es poco probable encontrarse con secuencias de palabras que se refieran a situaciones más actuales como *President Biden*. Además analizando los dos extractos el modelo parece haber relacionado *our nation* con *The United States* como se aprecia con la generación de entidades cercanas a los Estados Unidos. Cuando se realice el *fine-tuning* con los discursos de la reina de Inglaterra, esta característica no será probable, sino que relacionará la nación con Reino Unido o entidades similares.

... And as we have worked diligently to close the Afghan refugee crisis, the US Government has worked tirelessly to address the issues we have identified as potential future challenges for the Afghan people.

We now have an estimated 7 million Afghans living in the United States. ...

(Extracto del discurso A.1.1)

A medida que avanza el texto, parece que el hilo conductor se desvía del tema de la semilla dada: en este caso en este extracto se habla de la crisis de refugiados afganos (*Afghan refugee crisis*).

... Furthermore, the withdrawal of the United Kingdom from the European Union and to the Gulf states will allow us to bring more of our partners to participate in the fight against this pandemic. ...

... Finally, we are prepared to respond to a pandemic and other extreme events. The United States, as it was in 1996, will do so to defend our Nation and our allies, and to put an end to terrorism. ...

(Extractos del discurso A.1.1)

La segunda semilla intenta reconducir el hilo conductor (se ha adaptado el script de generación de texto ([run_generation.py](#)) para considerar como contexto el texto previamente generado). Debido a esto, se puede apreciar como el modelo vuelve a introducir la entidad “pandemia” (*pandemic*) y otras relacionadas. Sin embargo, se percibe que esta segunda semilla no aporta mucho más en el texto generado (no se consigue desviar el tema propuesto). Esto posiblemente se deba a que el conjunto de entrenamiento no posee

muchos ejemplos sobre pandemias.

Por otro lado, relaciona dicha pandemia con otros desastres como el terrorismo y la situación de Estados Unidos en 1996 (*United States, as it was in 1996 y terrorism*) (seguramente se refiere al atentado de Atlanta o el bombardeo de Irak).

... * * *

(CNN) President Barack Obama announced Friday that he had officially closed the U.S.-led coalition against the Ebola outbreak in West Africa.

The Obama administration will announce the closure of two main government-supported agencies to the agency that coordinates and coordinates the Ebola response in West Africa. ...

(Extractos del discurso A.1.1)

Tras varios párrafos hablando en primera persona, parece que hay un cambio de visión (también es significativa la marca de separación incluida entre estos párrafos (* * *)). Se puede intuir que extrae una cita de la CNN o que habla como si fuese dicho programa televisivo. Además, se vuelve a tratar el tema epidemiológico pero esta vez introduce el virus del Ébola (desvirtuando un poco el tema, ya que inicialmente se especificó que el virus era el COVID-19).

A continuación el texto se enfocará en la pandemia generada por el virus Ébola hasta el final de este, sin muchos rasgos interesantes más allá de los ya comentados.

Por lo general puede apreciar coherencia léxica, semántica y sintáctica durante todo el texto generado. Sin embargo, como es normal, pues no ha sido entrenado ni ajustado para ello, no se parece a un discurso, si no más bien a un artículo de opinión que toma características de una noticia en ciertas secciones del texto. Además se destaca que el contenido de dicho texto sigue gran parte del argumento introducido por las semillas (salvo la segunda, que no consigue aportar mucha información nueva), aunque se desvirtúe a medida que se avanza, perdiendo información de contexto.

Para obtener una visión más específica de algún resultado de partida se ha incluido en el anexo otro discurso generado con las mismas características (A.1.2). No se va a entrar en muchos detalles pues posee características muy similares al anterior descrito, influenciado por las tres semillas introducidas.

Cabe destacar que esta vez el texto generado toma el papel de una noticia sobre la vacuna del virus donde, además, se detallan consejos y recomendaciones sanitarias frente a este y a la posible vacuna.

A medida que avanza la lectura aparece, de nuevo, una mención al virus Ébola. Esto es

debido a que el brote que surgió en 2014 en África fue una de las enfermedades más actuales y documentadas de los últimos años, por lo que, posiblemente, el corpus de entrenamiento del modelo GPT-2 posea muchas referencias a dicha enfermedad y se relacione fácilmente con la semilla introducida.

Análogamente al anterior discurso, se hace referencia a Estados Unidos, concretamente al CDC (*Centers for Disease Control and Prevention*), una agencia del Departamento de Salud y Servicios Humanos de los Estados Unidos.

Con *fine-tuning*

Vista la situación de partida, se detallan a continuación las características de algunos discursos (y particularmente A.2.1) generados al realizar *fine-tuning* con el corpus de discursos de la reina de Inglaterra explicado en la sección 5.3.1. De nuevo se incluirán algunos extractos característicos de los discursos más interesantes de entre la totalidad de pruebas realizadas.

The pandemic generated by the COVID-19 virus has engulfed our nation since February of this year. The infectious disease continues to spread throughout Europe and it remains a major threat to human health in both developing and emerging nations.

During that time we have witnessed an unprecedented rise in mortality and serious illness, and an increasing tendency towards terrorism and armed conflict.

But while this continues, we must continue to recognise the importance of building a safe and accessible world for all people of every age and race.

I have no doubt that this country has successfully achieved this - and it was the task of my predecessors to ensure that every man and woman in our society was given the chance to know about and act on this important message of hope. ...

(Extracto del discurso A.2.1)

Desde el comienzo de este texto se puede apreciar que el estilo ha cambiado con respecto a los dos anteriores. Se hace uso constante de la primera persona, que es una característica fundamental en cualquier discurso.

Intuitivamente, aunque no se especifica, también se puede entender que la persona que habla es la reina de Inglaterra debido a que menciona a sus predecesores, lo que puede relacionarse con los antiguos monarcas (*my predecessors to ensure that every man and woman in our society*). También se trata como un mensaje de tranquilidad y esperanza para la sociedad (*we must continue to recognise the importance of building a safe and accessible world for all people of every age and race o act on this important message of hope*), mensajes que en los discursos anteriores no aparecían y que hacen pensar que es

realmente la reina la protagonista del discurso.

Por otra parte, como se puede comprobar, el modelo consigue de nuevo una correspondencia de entidades semánticas del campo de la medicina y de la epidemiología, dándole uso coherente y desarrollando el discurso con ellas. Además se vuelven a relacionar con desastres como el terrorismo y los conflictos armados.

... Furthermore, the withdrawal of the United Kingdom from the European Union is a significant defeat in our efforts to bring prosperity and prosperity to others. We need to take our message with the international community in its strong direction.

We can therefore continue to work together towards the end of this conflict and ensure the very best possible outcomes for the people of this great nation, and our children.

...

(Extracto del discurso A.2.1)

La incorporación de la segunda semilla provoca que el tema del discurso cambie de la epidemiología a la política, sin perder el sentimiento de comunidad y nación de los primeros párrafos, tal y como sería un discurso de un monarca.

... Madam President, it is this great strength of friendship, tolerance and goodwill which makes us so determined to live together in a peaceful, just and just world, and which will lead to a peaceful resolution of the international system of international law and order.” ...

(Extracto del discurso A.2.1)

Los siguientes párrafos son parecidos a este extracto. En ellos se hace mención a una presidenta (*Madam President*). Puede que haga referencia alguna de las dos expresidentas de Irlanda: Mary Robinson o Mary McAleese. Esto se debe a que el corpus de discursos del *fine-tuning* incluye algunos textos datados en los años 1970 y el corpus existen discursos donde la reina se refiere a ellas como *Madam President*. Resulta interesante comprobar como la narrativa del Brexit gira hacia Irlanda el tema del discurso.

... We are immersed in a period where our nation must get the best of itself and fight alongside other people who want to see us transformed into more democratic countries.

(Extracto del discurso A.2.1)

Por último, la última semilla le permite concluir el discurso, sin desvirtuar el tema de este durante toda la lectura. Se puede concluir, a pesar de tener fallos de contexto, que

el estilo es el de un discurso y que está relacionado correctamente con Inglaterra y con su monarca. En el anexo se incluye otro discurso generado con las mismas características (diríjese a A.2.2 para su lectura).

Sin más comentarios destacables salvo los siguientes: aparece la entidad “Ébola” que como se ha podido comprobar, es común encontrarla relacionada a las entidades “pandemia”, “virus”, etc.; se destaca también la aparición de entidades como “Parlamento” (*Parliament*), “Gran Bretaña” (*Britain*), la Mancomunidad de Naciones, fundada por el Parlamento de Reino Unido y con una aparición abundante en los discursos de la reina (*Commonwealth*); y el estilo es propio de un discurso a la sociedad.

Segunda inicialización de las semillas

Sin fine-tuning

Esta vez se han utilizado dos semillas para generar los siguientes textos: “*Today, I want to claim that we have an obligation to ensure a public education for all children*” y “*I’m sure the Prime Minister will do a great job*”.

Se incluyen, de nuevo, dos textos generados, a partir de dichas semillas, por el modelo GPT-2 original, sin *fine-tuning* con el corpus de discursos de la reina, para conocer la situación de partida: A.3.1 y A.3.2

No se va a realizar un análisis detallado (como el desarrollado con la primera inicialización) para estos textos, pues los resultados de interés son los generados una vez aplicado el *fine-tuning* y los resultados son similares a los de la primera inicialización. Sin embargo se destacan algunos recursos y objetos significativos.

Por un lado, es destacable el uso de *hashtag* para indicar la campaña *#IStandWithHer* del discurso A.3.1, que es un recurso lingüístico moderno que no había aparecido en los textos anteriores y se relaciona correctamente con la entidad “campaña” (*campaign*) como es normal en las redes sociales. Con esto se puede considerar que el modelo posee un amplio catálogo de recursos para desarrollar sus textos.

También se destaca que, con esta inicialización, los textos cambian completamente de tema respecto a los generados con la primera inicialización, incluyendo un léxico relacionado con el campo semántico de la educación, desarrollando el tema con entidades relacionadas a dicho campo de forma coherente, pero sin perder el estilo de discurso que se obtuvo al hacer *fine-tuning* con la primera inicialización.

Con fine-tuning

Vista la situación de partida, se detallan a continuación las características de algunos discursos generados al realizar *fine-tuning* con el corpus de discursos de la reina de Inglaterra explicado en la sección 5.3.1. De nuevo, se incluirán algunos extractos característicos

de los discursos más interesantes (en particular A.4.1) de entre la totalidad de pruebas realizadas.

Today, I want to claim that we have an obligation to ensure a public education for all children in Scotland.

We have a wide range of educational opportunities and opportunities, many of which are in the fields of science, technology and education. All of them require children to seek and accept real and fair opportunities. . .

(Extracto del discurso A.4.1)

El texto generado comienza por la primera semilla introducida y a partir de ella se conduce el discurso relacionando palabras del campo semántico de la educación y la ciencia. Además se hace uso durante todo el contenido de la primera persona y el receptor parece ser un colectivo, lo que provee de argumentos la idea de que se trate de un discurso.

... As I have told you last week, our work is about teaching young people how to be responsible, responsible, responsible and successful, whether in the job or in life itself. This will take courage and dedication from everyone.

I'm sure the Prime Minister will do a great job in that.

At the same time, I want to ensure that our children grow up to be selfless citizens. In that way, the future of Britain and of this country will look very different to how it looked at the time in 1945. . .

(Extracto del discurso A.4.1)

Destaca, de nuevo, el hecho de referirse a Gran Bretaña (“*Britain*”) en el discurso, influenciado por el *fine-tuning*.

... Many of you recall my speech to the Commonwealth Conference in 1995. I was pleased to witness how the Commonwealth was transformed from a simple organisation, with common interests in common issues and practical experience, into an innovative global organisation with strong values and aspirations. . .

(Extracto del discurso A.4.1)

Esta parte es característica pues se refiere a “su discurso de la Conferencia del Commonwealth en 1995” (*my speech to the Commonwealth Conference in 1995*). La reina de Inglaterra es la Jefa del Commonwealth lo que puede indicar que es la reina quién está hablando. El discurso continúa hasta el final siguiendo las mismas características, sin desvirtuar el tema.

Today, I want to claim that we have an obligation to ensure a public education for all children in the United Kingdom.

This has been my foremost aim for twenty years and I have a deep admiration for the progress we are making in the years since. I think all that we have achieved in the twenty-first century – education for all; science, technology, creative endeavour; and family life; – is testament to what has been achieved in the world's diverse and varied cities and towns. ...

(*Extracto del discurso A.4.2*)

El segundo ejemplo se desarrolla con características similares: claras referencias a Reino Unido, el estilo es el de un discurso cuyo emisor es una persona de la realeza o la política internacional, coherencia sintáctica y semántica desde la introducción, etc.

... On the occasion of this week, I welcome you all to the Assembly of Prince Philip for the first time. It will be a welcome opportunity for me to meet with you all, to give a well-reasoned presentation, and to look forward to a number of opportunities in the coming months. ...

(*Extracto del discurso A.4.2*)

Otra vez aparecen párrafos característicos, como el anterior, donde se hace mención a la “Asamblea del Príncipe Felipe” (*Assembly of Prince Philip*). Cabe destacar que el príncipe Felipe es el consorte de la reina de Inglaterra. Esto añade otra prueba de que el *fine-tuning* es efectivo.

5.3.3. Discusión

Uno de los desafíos en el desarrollo de sistemas de generación de texto es que no existe una métrica automatizada satisfactoria para evaluar el resultado final del sistema. Aún así, existen distintas métricas automatizadas de evaluación que son útiles para encontrar ejemplos en los que el rendimiento es deficiente y estas son consistentes para evaluar otros sistemas similares [58]: destacan ROUGE [60] y BLEU [61] (un conjunto de métricas que se utilizan de forma específica para evaluar tareas de resumen y traducción comparándolos con algún resumen o traducción, producidos por humanos); y la perplejidad, que es la que se usará como indicador de evaluación para el problema.

La perplejidad [59] de un modelo de lenguaje puede verse como el nivel de incertidumbre al predecir el siguiente token, donde, cuánto más baja sea, más confianza existe en la predicción de dicho token. Matemáticamente, la perplejidad se define como la probabilidad logarítmica promedio exponencial de una secuencia. Denotando por $x^{(i)}$ con $i \in \mathbb{N}$ a un

token del vocabulario \mathcal{V} , se toma una secuencia \mathcal{S} de m tokens $\{x^{(1)}, \dots, x^{(m)}\}$. Se calcula la perplejidad (PPL) de \mathcal{S} como:

$$PPL(\mathcal{S}) = \exp \left\{ -\frac{1}{m} \sum_{i=1}^m \log \left(P(x^{(i)} | x^{(1)}, \dots, x^{(i-1)}) \right) \right\}$$

donde $\log \left(P(x^{(i)} | x^{(1)}, \dots, x^{(i-1)}) \right)$ es la probabilidad logarítmica del i -ésimo token condicionada a tener los tokens anteriores.

Sabiendo esto, el modelo entrenado tiene una perplejidad de 20,77, que, dada la gran cantidad de tokens pertenecientes al vocabulario (50256 tokens), es un número bastante bueno. Esto se aprecia en la calidad de los textos generados. Son textos coherentes, correctos semántica y sintácticamente y consiguen una alta relación entre entidades similares.

5.4. Experimento 2

5.4.1. Diseño experimental

El segundo experimento realizado consiste en utilizar BERT para generar texto. Sin embargo, como se explicó en la sección 3.5.5, al ser un modelo bidireccional, en la práctica, las tareas generativas en las que destaca consisten en completar huecos en frases, traducir textos, etc., más orientadas a adivinar el contexto de una frase o a extraer *embeddings* que simplemente a generar texto continuo (como la tarea realizada en el experimento anterior por GPT-2). Por ello, la tarea elegida para realizar por el modelo será la de autocompletar frases. Sin embargo, cabe remarcar que en el paper publicado por Wang y Cho ([66]) se explica cómo, realizando algunos pequeños ajustes del modelo, se puede adaptar BERT para la tarea de generación de texto continuo.

Para la ejecución de esta tarea se hace uso del objeto *pipeline* proporcionado por la biblioteca Transformers [55]. Estos *pipelines* son objetos que abstraen la mayor parte del código complejo de la biblioteca, ofreciendo una API simple dedicada a varias tareas, entre la que se incluye la que se pretende realizar.

Por otro lado, aprovechando que el modelo BERT funciona bastante bien con texto en idiomas diferentes al inglés (la bidireccionalidad es más adecuada para idiomas con frases largas o que no usan un orden gramatical más estricto como el inglés), usamos la versión del modelo [dccuchile/bert-base-spanish-wwm-cased](#): un modelo con la arquitectura entrenado en PyTorch con textos en español con el objetivo de minimizar el error provocado por la tarea de modelado de lenguaje enmascarado (*masked language modeling*, MLM) [25]. Esta tarea consiste en elegir muestras al azar (de forma natural se eligen muestras que forman el 15 % del total) de tokens del corpus y reemplazarlos por el token **[MASK]** (con

probabilidad 0,8), por un token aleatorio (con probabilidad 0,1) o mantenerlos invariantes (con probabilidad 0,1).

Puede visitar el [código](#) para visualizar cómo se ha desarrollado la tarea explicada.

5.4.2. Resultados

A continuación se van a incluir algunos ejemplos de frases a llenar junto con las 5 posibles respuestas más probables del modelo. El hueco a llenar será aquel que incluya el token **[MASK]** y el modelo devolverá los 5 tokens más probables con su porcentaje de probabilidad. Se ha transformado la salida del modelo en una tabla para una mejor visualización.

Frase original	Token predicho	Valor de la predicción
HuggingFace es una [MASK] que la comunidad usa para resolver tareas de NLP.	herramienta	0.3751893
	aplicación	0.2434827
	función	0.0623801
	interfaz	0.0456741
	utilidad	0.0290186

Este primer ejemplo indica que para la frase *HuggingFace es una [MASK] que la comunidad usa para resolver tareas de NLP.* el modelo predice que la palabra (token) que mejor se acopla a dicha frase (sustituyendo al token **[MASK]**) es *herramienta* con un 0,3751893 de probabilidad. Sin embargo también proporciona otras alternativas como *aplicación* con un 0,2434827 de probabilidad, *función* con un 0,0623801, *interfaz* con un 0,0456741 y *utilidad* con un 0,0290186. Como se puede apreciar, existen otros tokens que tienen probabilidad de aparecer (pues la suma de probabilidades no es 1), aunque aquí sólo se muestran los más probables.

Se puede apreciar coherencia sintáctica, pues se espera que detrás de un determinante vaya un sustantivo y todos los tokens predichos más probables son sustantivos. Además, también se aprecia coherencia semántica y léxica, pues relaciona una biblioteca de programación que resuelve tareas con una “herramienta”, “aplicación”, etc. que en definitiva son elementos del mismo campo semántico.

Frase original	Token predicho	Valor de la predicción
El coronavirus es un virus que ha [MASK] a muchas personas.	afectado	0.5722662
	matado	0.3109934
	confundido	0.0184451
	sorprendido	0.0138619
	atacado	0.0081479

El segundo ejemplo es parecido al anterior. En este caso se espera un verbo (concretamente un participio) y, de forma correcta, es lo que el modelo predice. Además, se le da una alta probabilidad a dos palabras frente a todas las demás: *afectado* y *matado* que, intuitivamente es lo que se espera que haga un virus con las personas.

Frase original	Token predicho	Valor de la predicción
Si dos más tres es cinco, cuatro más dos es [MASK]	seis	0.2980819
	cuatro	0.1765361
	siete	0.0776952
	cinco	0.0716560
	diez	0.0643907

Exprimiendo un poco la lógica del modelo, surgió la duda de saber como se comportaba frente a problemas de relación. En este tercer ejemplo se pone a prueba el modelo con una suma. El primer punto a destacar es que predice tokens que son números, que realmente es lo esperable (en otros ejemplos análogos, que se pueden visualizar en el [código](#), predice palabras como “suficiente” y otros adjetivos o adverbios de cantidad, que también son esperables). También es destacable que acierte con la suma pues indica que “cuatro más dos” es “seis”, lo que matemáticamente es correcto (si se considera el álgebra usual). Sin embargo, la probabilidad de predicción de dicho token no es elevada por lo que la predicción no es contundente.

Frase original	Token predicho	Valor de la predicción
Si uno más uno es dos y uno más dos es tres, uno más tres es [MASK].	cuatro	0.3159345
	uno	0.2552272
	dos	0.0795350
	tres	0.0500085
	cinco	0.0467583

De forma similar al ejemplo anterior, se pone a prueba la lógica matemática: en este caso se construye un contexto más amplio para dotar al modelo de mayor información e intentar así que la predicción sea más contundente. Sin embargo como se puede apreciar, aunque el resultado es correcto, la diferencia de probabilidad con respecto a los demás tokens no es abultada, por lo que se intuye que el modelo no es capaz de establecer una relación lógica matemática correcta. Como ejemplo de este razonamiento se incluye el siguiente ejemplo:

Frase original	Token predicho	Valor de la predicción
Si dos por tres es seis entonces cinco por dos es [MASK].	seis	0.1817645
	siete	0.1644815
	cinco	0.1198562
	ocho	0.0690614
	cuatro	0.0626170

En este ejemplo el resultado no es el esperado, de hecho el resultado esperado (“diez”) no aparece ni entre los tokens probables. El modelo es capaz de entender que el token esperado pertenece al campo semántico de los números, pero no establece la relación lógica para predecir el correcto.

Frase original	Token predicho	Valor de la predicción
Antonio es mi padre, por lo que yo soy el [MASK] de Antonio.	padre	0.6495881
	hijo	0.1508793
	hermano	0.0891699
	tío	0.0152973
	cinco	0.0144875

Como último ejemplo se propone una relación en el campo semántico de la familia. Como se puede observar, el modelo consigue seleccionar tokens de dicho campo semántico pero no consigue acertar con el token esperado (“hijo”), siguiendo la relación lógica de la frase.

5.4.3. Discusión

Por lo anteriormente expuesto, se considera que el modelo BERT, entrenado de forma correcta, es capaz de establecer relaciones entre los campos semánticos, mantener coherencia léxica, sintáctica y semántica y tiene proyección para realizar tareas en el campo del *Natural Language Processing*.

Se ha observado que no es capaz de obtener lógica relacional en algunos aspectos, pero se espera que se consiga solventar dichos problemas entrenándolo con corpus adecuados a la tarea que se pretende resolver, y no un corpus genérico como en el ejemplo mostrado.

Los resultados obtenidos, por tanto, se consideran notables.

Todo el proyecto está disponible en: <https://github.com/AlbertoEstep>.

Capítulo 6

Conclusiones y trabajo futuro

Como parte final del trabajo se van a exponer las diferentes conclusiones obtenidas a lo largo del desarrollo del mismo. Asimismo se dejan algunas propuestas y trabajos futuros relevantes para continuar con esta investigación.

6.1. Conclusiones

En estas páginas se ha intentado mostrar que los modelos generativos del lenguaje forman una herramienta muy potente y de gran proyección para desarrollar soluciones tecnológicas para abordar una alta cantidad de problemas del mundo real.

La distancia en la comunicación entre personas y computadoras se está reduciendo a medida que la investigación y la evolución en el campo del *Natural Language Processing* avanza.

El gran número de soluciones y tareas que son (y serán) capaces de resolver estos modelos abre una infinidad de posibilidades para desarrollar tecnología que haga más fácil la vida humana. Entre las muchas posibilidades que se podrán conseguir en tiempo cercano se destaca, por ejemplo, el poder aumentar la abstracción en la tarea de programación simplemente usando lenguaje natural, lo que puede acercar a la gran mayoría de la población a la programación y facilitar el trabajo de los programadores.

En este trabajo se ha profundizado en los aspectos teóricos de los modelos del lenguaje más actuales, partiendo de una base sencilla y avanzando a medida que pasan las secciones. En concreto se ha realizado un estudio progresivo, desde las redes neuronales sencillas, hasta las estructuras neuronales que forman los pilares de los modelos más avanzados (estas estructuras estudiadas son las redes neuronales recurrentes y los *transformers*, con sus distintas variantes y adaptaciones). De forma paralela se ha profundizado en los desarrollos matemáticos que permiten la optimización y entrenamiento de estas

arquitecturas neuronales, profundizando en las funciones de pérdida y en los métodos de optimización.

Como último aspecto, se ha podido comprobar con ejemplos sencillos cómo, con un poco de esfuerzo, estos modelos resuelven las tareas del lenguaje que les son propuestas. En particular se han comprobado las capacidades de dos modelos (GPT-2 y BERT) para las tareas de generación de texto y completación de oraciones, obteniendo resultados notables.

6.2. Trabajo futuro

La investigación y evolución del área del *Natural Language Processing* avanza a pasos agigantados. Como ejemplo de esto, a la vez que se realizaba la experimentación de este proyecto, la biblioteca con la que se ha trabajado (véase sección 5.2) ha estado continuamente actualizándose (incluso ha habido cambios en los ficheros se estaban usando para la realización de los experimentos).

Otro ejemplo de dicha evolución es el modelo GPT-3, que fue lanzado en mayo de 2020 y actualmente no está disponible para la comunidad. Este hecho ocurrió también con el modelo GPT-2, que inicialmente fue censurado a la comunidad, publicando solo una versión más simple, y no fue hasta unos meses más tarde cuando se publicó la versión completa. Como trabajo futuro y muy interesante se propone realizar una investigación práctica de las capacidades de dicho modelo (GPT-3) cuando esté disponible ya que se presupone muy superior a cualquiera actualmente publicado.

Otro aspecto interesante es que la gran mayoría de investigación y práctica con los modelos generativos del lenguaje están basados en entrenamiento y corpus en inglés (sobre todo si se habla de generación continua de texto). Un trabajo futuro que se propone es investigar y llevar a cabo un desarrollo de los modelos generativos capaces de generar texto en español de forma mucho más eficiente y correcta.

El anterior punto también lleva a la propuesta de realizar un estudio comparativo más profundo de realización de tareas en español con respecto al inglés, por ejemplo entrenando un modelo GPT con muchos más textos en español.

Por último, tras la fama de las redes generativas adversarias (GANs) en otros campos del *deep learning*, como la Visión por Computador, surge la idea de aplicar dichas redes generativas en el campo del *Natural Language Processing*. Para ello se dispone de algún *framework* como TextGan [67]. Se propone ampliar los contenidos de este proyecto, tanto teóricos como prácticos, investigando sobre la aplicación de las GANs en la tarea de generación de texto.

Anexo

Apéndice A

Discursos de la reina de Inglaterra

A.1. Discursos sin *fine-tuning*, primera inicialización

A.1.1. Discurso 1

The pandemic generated by the COVID-19 virus has engulfed our nation since February of this year, which triggered an enormous surge in emergency preparedness, health care, disaster relief and medical treatment. It is now happening again in Afghanistan and elsewhere.

The United States has put billions in emergency health funding in Afghanistan, and is now able to mobilize the United States Government and the international community to fight this epidemic. The humanitarian response of the United States Government to the epidemic has been great. But it has been far too little, far too late.

President Obama has the resources and will to provide these assistance to our nation, our partners, and all the world's people. Yet in an effort to do that, he has allowed an enormous financial, political and psychological advantage to the US Government. This advantage has been brought to the United States of America through the COVID virus.

And as we have worked diligently to close the Afghan refugee crisis, the US Government has worked tirelessly to address the issues we have identified as potential future challenges for the Afghan people.

We now have an estimated 7 million Afghans living in the United States.

So, we are prepared to respond to the pandemic and our country is prepared to face

this threat with urgency and force.

Furthermore, the withdrawal of the United Kingdom from the European Union and to the Gulf states will allow us to bring more of our partners to participate in the fight against this pandemic.

This includes the United States government and many others.

This includes both our local partners and our allies in Europe.

Finally, we are prepared to respond to a pandemic and other extreme events. The United States, as it was in 1996, will do so to defend our Nation and our allies, and to put an end to terrorism.

As we approach the final round of negotiations to extend the agreement between our Government and its partners for the continued support and protection of Afghanistan and other nations, we must remember that we have to act with common ground and stand ready to act with any additional partners.

For us, the two most important and important elements are peace, justice and security.

We must maintain that alliance and that partnership through all its possible means.

And if peace does not come soon, then we must protect our Nation.

And this is where we must all get together and take on this epidemic.

* * *

(CNN) President Barack Obama announced Friday that he had officially closed the U.S.-led coalition against the Ebola outbreak in West Africa.

The Obama administration will announce the closure of two main government-supported agencies to the agency that coordinates and coordinates the Ebola response in West Africa.

We are immersed in a period where our nation must get the best of itself and fight the Ebola epidemic, in an effort to keep our Nation healthy and keep our communities safe.

The decision to close government agencies was announced Friday, amid heavy public pressure from our nation's public servants and international organizations as we speak about our response.

There are two main reasons why we closed the agency that coordinates and coordinates the Ebola response:

1) We had to help bring in the government experts we needed to prevent and respond to the epidemic in West Africa and in Africa as it developed.

In this critical time, we need help. We must do so in collaboration with partners and allies from outside the United States and Europe.

When we were in West Africa, the United States had a limited understanding of the outbreak and the government response, and were increasingly frustrated with the state of health in Guinea and Liberia.

Now we must improve our health system in response to the Ebola outbreak.

In addition, we must move beyond an assumption of responsibility to a national strategy to keep the government focused on its mission and keeping our country's health service in good standing.

The national strategy is an opportunity to better protect our Nation from the threat of Ebola by focusing on those who have a significant impact in developing and protecting our health systems, including the WHO.

(Discurso 1, generado por el modelo GPT-2 sin *fine-tuning*)

A.1.2. Discurso 2

The pandemic generated by the COVID-19 virus has engulfed our nation since February of this year. We can anticipate that by November 1, our national health department will not be able to provide timely and effective vaccine. Therefore, we now offer one solution to alleviate all of this uncertainty: vaccine access. The latest Vaccine Access Fund proposal states that only licensed immunization clinics can receive the vaccine with the recommended vaccine dose (Rv-1) and that the health care provider who collects and delivers the vaccine (i.e., school district or hospital district) must certify that they have received all of the vaccines recommended. Thus, all of the primary vaccine manufacturers cannot be found on the list and therefore can only meet with school health care providers. To ensure that vaccines are provided in the same quantity and form as approved vaccines, the Vaccine Access Fund has partnered with our local vaccination providers to create and fund an updated National Childhood Vaccine Injury (NCVITI) program. (See Vaccine Coverage)

If your child is at risk of contracting a contagious virus, the CDC provides an updated CDC Vaccine Injury Information sheet.

If you or someone you know has recently contracted an Ebola virus, you can ask your

child for specific information on a CDC Vaccine Injury Information sheet that can help you assess vaccine effectiveness. You can also learn about what vaccine treatments have been recommended for children who have contracted an Ebola virus. For more information about the National Childhood Vaccine Injury (NCVITI) program, visit the CDC's National Childhood Vaccine Injury Program webpage.

Furthermore, the withdrawal of the United Kingdom from the European Union's Children's Vaccine Programme is an important step that will make vaccines available in some circumstances even if they are not available to all European countries. The withdrawal of the United Kingdom from the European Union's Children's Vaccine Programme also addresses concerns about vaccine safety for children in developing countries. A number of countries have implemented precautionary measures to prevent such outbreaks.

Your child can use the CDC Vaccine Injury Information Sheet to follow the steps outlined below:

Ensure you:

Are at risk of contracting a contagious virus.

Are aware of any recommended vaccines and vaccines have been administered to you or are in a vaccine-preventive setting.

Do not give your child too much information, for example, because the information might include what is prescribed.

Prepare a copy of the CDC Vaccine Injury Information Sheet.

Use the Vaccine Injury Information Sheet on your child's home computer.

Remember: If your child's health care provider cannot provide the vaccine that he or she required and can't provide the recommended dose, you should be prepared to provide that information to that health care provider in writing by telephone or e-mail.

Fully vaccinate your child (i.e., take it orally and use one dose every 7 days) for two years after the person leaves the state. For more information about vaccinations available to those at risk of contracting Ebola, contact CDC at (800) 825-1033.

We are immersed in a period where our nation must get the best of itself and fight for the protection of children's health and the country's future. If we are not at risk of contracting Ebola, we will make sure we take appropriate precautions. CDC is dedicated to providing the best vaccine available, whether we have it or not. We will do everything possible to keep the public safe by conducting medical and medical education, and the work of the CDC is ongoing.

(Discurso 2, generado por el modelo GPT-2 sin *fine-tuning*)

A.2. Discursos con *fine-tuning*, primera inicialización

A.2.1. Discurso 3

The pandemic generated by the COVID-19 virus has engulfed our nation since February of this year. The infectious disease continues to spread throughout Europe and it remains a major threat to human health in both developing and emerging nations.

During that time we have witnessed an unprecedented rise in mortality and serious illness, and an increasing tendency towards terrorism and armed conflict.

But while this continues, we must continue to recognise the importance of building a safe and accessible world for all people of every age and race.

I have no doubt that this country has successfully achieved this - and it was the task of my predecessors to ensure that every man and woman in our society was given the chance to know about and act on this important message of hope.

But what must we remember when we look back, for instance, to today?

Looking at this story from the other side, a century ago, we have lost the power of history. So much so that we have been unable to reach agreement on our historic relationship for much longer than that.

To be clear, we are now closer together. The challenges ahead will not be the same as they were in the 1920s or '30s, when we met in Vienna and a generation ago were on their first successful joint visit to Great Britain.

There are many important differences in my view between our two countries. But that does not mean that we are unable to work through them together, for we share in their prosperity, shared values and hopes.

Furthermore, the withdrawal of the United Kingdom from the European Union is a significant defeat in our efforts to bring prosperity and prosperity to others. We need to take our message with the international community in its strong direction.

We can therefore continue to work together towards the end of this conflict and ensure the very best possible outcomes for the people of this great nation, and our children.

Madam President, it is this great strength of friendship, tolerance and goodwill which makes us so determined to live together in a peaceful, just and just world, and which will lead to a peaceful resolution of the international system of international law and order.”

Sisters, as I have done before, have taken great pride in their parents and grandparents and all their support throughout this troubled period of time.

I wish you all a very happy and exciting journey, for as much as I can, there are some things you may not be able to do at this moment.

Madam President, I congratulate you on this evening's address and wish you all a very happy and enjoyable month. I look forward to talking to you again on the occasion of your return from St. Petersburg.

Madam President, your two visits to Australia and New Zealand give us the opportunity to reflect on our shared interests and what is at stake in achieving our common goals.

We are immersed in a period where our nation must get the best of itself and fight alongside other people who want to see us transformed into more democratic countries.

(Discurso 3, generado por el modelo GPT-2 con *fine-tuning*)

A.2.2. Discurso 4

The pandemic generated by the COVID-19 virus has engulfed our nation since February of this year.

The Ebola outbreak in West Africa has left many thousands of people homeless. The Government has put hundreds of refugees into emergency shelters, while giving humanitarian aid.

As the year draws to a close I am reminded of the humanitarian responsibility of our Government and the dedication of our Parliament to the great service it provides to our people.

Furthermore, the withdrawal of the United Kingdom from the European Union has already caused considerable disruption to the economic and financial life of the United Kingdom.

Over the last twelve months I have met with representatives of these governments from around the world to discuss issues such as the Millennium Development Goals, the World Trade Organisation and the Trade in Services Agreement.

Throughout this visit, I have been struck by the warmth of the people in Britain and throughout the Commonwealth who have been touched by the messages of unity, prosperity and peace. They are looking for the strength in which to remain in the Commonwealth and help those around the world continue to thrive.

We are immersed in a period where our nation must get the best of itself and fight to retain its leadership, which in turn can enable the Commonwealth to flourish. We have the potential to lead the world, and in particular to maintain the peace and security of our seas and communities - whatever the outcome of this engagement.

In my lifetime, I have watched as Europe's nations have been confronted by new challenges and uncertainties, from the rise of terrorism to the climate change of the last few years. At times, however, we have responded well by increasing our focus and courage.

The opportunities we have face many of these challenges with little or no guidance from outside, as they present an opportunity for our own success, or for others to find, through an inclusive process of government, to sustain the trust of our people and for the prosperity and security of the whole of Europe.

I am confident that the Commonwealth can serve to guide the direction of good governance and to ensure a brighter future for our peoples. But this alone should not, of course, serve to guide our actions alone.

For the future, I hope that we will work closely together. And the Commonwealth must remain relevant to the future and to the future of our governments and to the future of the lives of its citizens. But in the meantime, we must build and nurture our own future in a way that is not only honest but also sustainable.

(Discurso 4, generado por el modelo GPT-2 con *fine-tuning*)

A.3. Discursos sin *fine-tuning*, segunda inicialización

A.3.1. Discurso 5

Today, I want to claim that we have an obligation to ensure a public education for all children. That's why I'm launching #IStandWithHer campaign to push for an education policy that recognizes that we need the opportunity to grow our communities.

I've introduced the first state education law in more than 40 years. It will make it easier for every child in every community to attend a college. I've supported all of our states' efforts to develop the most comprehensive and effective public education system in the country.

Since our founding in 1978, we have spent years working to deliver quality, secure, safe, and affordable public education.

I also have a history of working to address America's racial and ethnic disparities.

The first federal district court judge in the country sentenced my brother and sister to 25 years in prison for drug trafficking. They had pleaded guilty to conspiring to produce child pornography, while my sister has been serving a sentence of 20 years in prison for possessing child pornography.

I support the victims and all victims of sexual assault: I support criminal justice reforms, but I know that's going to cost an awful lot.

What's next? I want to take another step forward and pass a \$2 billion bill, to pay for all of the necessary capital improvements and to address rising tuition. I want to address a broken federal and state education system that doesn't do enough to improve quality and provide the right education for all.

I'm sure the Prime Minister will do a great job doing that. We're going to do just about anything to keep education open and accessible.

I'll vote for the education bill at the next government meeting. I'm also calling on everyone in the country to join me in passing a bipartisan bill to provide the right education for all.

We need the right teachers for America, and we can't just let this happen. We need teachers to teach kids as much as possible.

And I believe in a fair system of government for all of us. And I'm proud that I have the support of this party. And we need to do our part to protect the rights of all Americans and ensure the health and safety of all Americans.

I'm proud that we're able to raise the wages of teachers in our country.

So let's raise wages — we don't just work to raise wages, we actually strengthen the rule of law.

We've had workers who've been on the payroll for over a decade who have won't be out of school, who have paid their salaries — and they're getting great educations in this country, so we're going to do something about it.

And I support this new American initiative called #IStandWithHer, and I believe it's important to get the right education for every American, regardless of class.

(Discurso 5, generado por el modelo GPT-2 sin *fine-tuning*)

A.3.2. Discurso 6

Today, I want to claim that we have an obligation to ensure a public education for all children to succeed.

My current goal is to achieve our first public school voucher program by 2020.

As I campaigned, I began to introduce public policy to help our kids thrive as first-graders and as students. As I made the transition from university to private school in 2013, I was joined in that commitment by a great many parents of students.

But the charter school experience is different. It provides much better results for kids who are poor, and for students in a higher cost bracket. The costs of the charter program are high, often exceeding what it cost to build a private school to support these students.

As I talk more about what the voucher system needs to address, we must look at some possible solutions.

- For everyone else, I urge you to see, as I've said, why we should hold the public responsible for what we've done.

I know that this campaign will raise many questions. What about our policies, our curriculum, how our schools teach, how our students learn, the impact on families that this can have on our communities?

Here is what I hope we can learn, and what we can work toward, after this election.

I'm sure the Prime Minister will do a great job here.

But the facts don't matter. And I'd like to encourage the Prime Minister, as well, to talk to my opponents and ask them about the issues that matter to their children and their families.

I also want to make sure that our Government is working on reforms that make our children's education more equitable.

That means giving more money to teachers, increasing charter schools, ending financial and administrative barriers for parents and ensuring more resources are available to help our children achieve the best possible education.

So it is time to start this debate.

The time has come to get real about a change in our approach to our public schools.

The time has come to restore some of our core values.

So today, my government would like to take one step forward, step by step, to rebuild a public schools system that is the best for our kids.

To do that, we will bring all children together in a safe and sustainable community

and build upon the most important principles of public education that they've heard before.

I hope to have you join me as I speak, and hear from people who have received vouchers and that I want to make clear that I can't wait to move forward.

If you believe that you are an important voice in the charter debate, then please sign this petition.

(Discurso 6, generado por el modelo GPT-2 sin *fine-tuning*)

A.4. Discursos con *fine-tuning*, segunda inicialización

A.4.1. Discurso 7

Today, I want to claim that we have an obligation to ensure a public education for all children in Scotland.

We have a wide range of educational opportunities and opportunities, many of which are in the fields of science, technology and education. All of them require children to seek and accept real and fair opportunities.

My administration is committed to building on the achievements of the last few years to ensure that this exciting future can be enjoyed at the same time as supporting the best in the lives of our children.

Over the past eight years we have brought people from every generation to the fore in the fields of education and science.

As I have told you last week, our work is about teaching young people how to be responsible, responsible, responsible and successful, whether in the job or in life itself. This will take courage and dedication from everyone.

I'm sure the Prime Minister will do a great job in that.

At the same time, I want to ensure that our children grow up to be selfless citizens. In that way, the future of Britain and of this country will look very different to how it looked at the time in 1945.

Many of you recall my speech to the Commonwealth Conference in 1995. I was pleased to witness how the Commonwealth was transformed from a simple organisation, with common interests in common issues and practical experience, into an innovative global organisation with strong values and aspirations.

Britain is on the cusp of a remarkable transformation, indeed, of its nationhood. It

can and must work hard and successfully to meet challenges in a global economy, to ensure that global business and industry flourish and prosper, and to fight poverty and inequality.

In the years ahead, and in the years ahead, we will continue to build on the Commonwealth values that have made Britain the world's economic power and leading global financial centre. But we will also work together in good faith and not only through cooperation and in common interests.

This can be achieved in so many ways, but the way things work will become much easier if we work together in common cause, and at the same time in the spirit of the spirit of friendship and consensus.

There are three main themes which will make this summit of the Commonwealth: the United Kingdom, the democratic institutions of the Commonwealth and the United Nations.

(Discurso 7, generado por el modelo GPT-2 con *fine-tuning*)

A.4.2. Discurso 8

Today, I want to claim that we have an obligation to ensure a public education for all children in the United Kingdom.

This has been my foremost aim for twenty years and I have a deep admiration for the progress we are making in the years since. I think all that we have achieved in the twenty-first century – education for all; science, technology, creative endeavour; and family life; – is testament to what has been achieved in the world's diverse and varied cities and towns.

As we face challenges in international communities and elsewhere, and as we travel, we must always recognise the immense work that remains to be done to improve opportunities for all.

On the occasion of this week, I welcome you all to the Assembly of Prince Philip for the first time. It will be a welcome opportunity for me to meet with you all, to give a well-reasoned presentation, and to look forward to a number of opportunities in the coming months.

Your visits to the Netherlands and the United Kingdom will surely have a significant impact on my thinking about the Commonwealth as we shape the future. More than ever, there is an emphasis on national pride and of mutual concern for the future of society.

I'm sure the Prime Minister will do a great job of conveying that message, and I look forward to welcoming you all on that great, historic and national stage.

(Discurso 8, generado por el modelo GPT-2 con *fine-tuning*)

Bibliografía

- [1] IQBAL & QURESHIA, *The survey: Text generation models in deep learning.* Journal of King Saud University - Computer and Information Sciences. (2020).
- [2] HUTCHINS, *From first conception to first demonstration: the nascent years of machine translation, 1947-1954. A chronology.* Machine Translation 12, 195-252. (1997).
- [3] STAHLBERG, *Neural Machine Translation: A Review.* Journal of Artificial Intelligent Research, vol 69, arXiv:1912.02047. (2019).
- [4] QUESADA, CALLEJAS & GRIOL, *Informe sobre los sistemas conversacionales multimodales multilingües.* Tecnologías y arquitecturas para el desarrollo de asistentes virtuales, sistemas de diálogo y otros interfaces conversacionales (2019).
- [5] MITRA & CRASWELL, *An Introduction to Neural Information Retrieval.* Foundations and Trends® in Information Retrieval, 13, 1-126 (2018).
- [6] LECUN, BENGIO & HINTON, *Deep Learning.* Nature, 521, 436-444 (2015).
- [7] ABU-MOSTAFA, MAGDON-ISMAIL & LIN, *Learning from data.* AMLbook.com (2012) ISBN 10:1-60049-006-9, ISBN 13:978-1-60049-006-4.
- [8] BENGIO, DUCHARME, VINCENT & JANVIN, *A neural probabilistic language model.* J. Mach. Learn. Res., 3:1137-1155 (2003).
- [9] CORRADO, MIKOLOV, CHEN & DEAN, *Efficient estimation of word representations in vector space.* arXiv:1301.3781v3, (2013).
- [10] PENNINGTON, SOCHER & MANNING, *GloVe: Global Vectors for Word Representation.* Empirical Methods in Natural Language Processing (EMNLP), 1532-1543, (2014).
- [11] RUMELHART, HINTON & WILLIAMS, *Learning representations by back-propagating errors.* Nature, 323(9), 533-536 (1986).
- [12] IOFFE & SZEGEDY, *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.* arXiv:1502.03167v3 [cs.LG] (2015).
- [13] SRIVASTAVA, HINTON, KRIZHEVSKY, SUTSKEVER & SALAKHUTDINOV, *Dropout: A*

- Simple Way to Prevent Neural Networks from Overfitting.* Journal of Machine Learning Research, 15, 56, 1929-1958 (2014).
- [14] KROGH & HERTZ, *A Simple Weight Decay Can Improve Generalization.* Advances in Neural Information Processing Systems 4, 950-957 (1992).
- [15] NG, KATANFOROOSH & MOURRI, *Sequence Models.* Curso Coursera, deeplearning.ai (2020).
- [16] CHEN, *A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation.* arXiv:1610.02583v3 [cs.LG] (2018).
- [17] HOCHREITER & SCHMIDHUBER, *Long Short-term Memory.* Neural computation, 9, 1735-1780, 10.1162/neco.1997.9.8.1735 (1997).
- [18] CHO, VAN MERRIËNBOER, GULCEHRE, BAHDANAU, BOUGARES, SCHWENK, & BENGIO, *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation.* Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1724-1734, 10.3115/v1/D14-1179 (2014).
- [19] SUTSKEVER, VINYALS & LE, *Sequence to Sequence Learning with Neural Networks.* arXiv:1409.3215v3 [cs.CL] (2014).
- [20] PETERS, AMMAR, BHAGAVATULA & POWER *Semi-supervised sequence tagging with bidirectional language models,* CoRR, abs/1705.00108, arXiv:1705.00108 (2017).
- [21] VASWANI, SHAZER, PARMAR, USZKOREIT, JONES, GOMEZ, KAISER & POLOSUKHIN *Attention Is All You Need* CoRR, arXiv:1706.03762 (2017).
- [22] PETERS, NEUMANN, IYYER, GARDNER, CLARK, LEE & ZETTLEMOYER *Deep Contextualized Word Representations* Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), 2227-2237 arXiv:1705.00108 (2017).
- [23] HOWARD & RUDER *Universal Language Model Fine-tuning for Text Classification* Association for Computational Linguistics (ACL) <http://arxiv.org/abs/1801.06146> (2018).
- [24] RADFORD, NARASIMHAN, SALIMANS & SUTSKEVER *Improving Language Understanding by Generative Pre-Training* OpenAI (2018).
- [25] DEVLIN, CHANG, LEE & TOUTANOVA *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 4171-4186 (2019).
- [26] RADFORD, WU, CHILD, LUAN, AMODEI & SUTSKEVER *Language Models are Un-*

- supervised Multitask Learners* OpenAI (2019).
- [27] BROWN, MANN, RYDER, SUBBIAH, KAPLAN, DHARIWAL, NEELAKANTAN, SHYAM, SAstry, ASKELL, AGARWAL, HERBERT-VOSS, KRUEGER, HENIGHAN, CHILD, RAMESH, ZIEGLER, WU, WINTER & AMODEI *Language Models are Few-Shot Learners* OpenAI (2020).
- [28] DAI, YANG, YANG, CARBONELL, LE & SALAKHUTDINOV *Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context* CoRR arXiv:1901.02860 (2019).
- [29] RAE, POTAPENKO, JAYAKUMAR & LILLICRA *Compressive Transformers for Long-Range Sequence Modelling* arXiv:1911.05507 (2019).
- [30] CHILD, GRAY, RADFORD & SUTSKEVER *Generating Long Sequences with Sparse Transformers* CoRR arXiv:1904.10509 (2019).
- [31] YANG, DAI, YANG, CARBONELL, SALAKHUTDINOV & LE *XLNet: Generalized Autoregressive Pretraining for Language Understanding* Curran Associates, Inc., Advances in Neural Information Processing Systems 32, 5753-5763 (2019).
- [32] SUN, WANG, LI, FENG, CHEN, ZHANG, TIAN, ZHU, TIAN & WU *ERNIE: Enhanced Representation through Knowledge Integration* CoRR arXiv:1904.09223 (2019).
- [33] SUN, WANG, LI, FENG, TIAN, WU & WANG *ERNIE 2.0: A Continual Pre-training Framework for Language Understanding* CoRR arXiv:1907.12412 (2019).
- [34] ROSSET *Turing-NLG: A 17-billion-parameter language model by Microsoft* Microsoft Research Blog (2020).
- [35] QIAN *On the Momentum Term in Gradient Descent Learning Algorithms* Neural networks: the official journal of the International Neural Network Society, 12 1, 145-151 (1999).
- [36] NESTEROV *A method for unconstrained convex minimization problem with the rate of convergence $o(\frac{1}{k^2})$* . Doklady ANSSSR (translated as Soviet.Math.Docl.), 269, 543-547 (1983).
- [37] DUCHI, HAZAN & SINGER *Adaptive Subgradient Methods for Online Learning and Stochastic Optimization* J. Mach. Learn. Res., 12, 2121-2159 (2011).
- [38] ZEILER *ADADELTA: An Adaptive Learning Rate Method* CoRR, arXiv:1212.5701 (2012).
- [39] IGEL & HÜSKEN *Improving the Rprop Learning Algorithm* Proceedings of the Second International Symposium on Neural Computation, NC'2000, 115-121, ICSC Academic Press (2000).
- [40] HINTON, SRIVASTAVA & SWERSKY *RMSprop: Divide the gradient by a running Ave-*

- rage of its recent magnitude* COURSERA: Neural Networks for Machine Learning. Lecture 6e (2012).
- [41] KINGMA & BA *Adam: A Method for Stochastic Optimization* arxiv:1412.6980, International Conference for Learning Representations, San Diego (2014).
- [42] DOZAT *Incorporating Nesterov Momentum into Adam* Workshop track - ICLR (2016).
- [43] RUDER *An overview of gradient descent optimization algorithms* arxiv:1609.04747, versión 2017 (2016).
- [44] MANNING & LAMM *CS224n: Natural Language Processing with Deep Learning* Stanford University, [CS224n](#) (2020).
- [45] Ng *Deep Learning Specialization* Coursera, deeplearning.ai [Curso](#) (2020).
- [46] OLAH Colah: *Understanding LSTM Networks* [Blog Colah](#) (2015).
- [47] CHOLLET *Transfer learning & fine-tuning* [Documentación de Keras](#) (2020).
- [48] RONG *Word2Vec Parameter Learning Explained* arxiv:1411.2738 (2014).
- [49] CYBENKO *Approximation by superpositions of a sigmoidal function* Mathematics of Control, Signals, and Systems (MCSS), 303-314, 2, doi:10.1007/BF02551274 (1989).
- [50] LESHNO, LIN, PINKUS & SCHOCKEN *Multilayer feedforward networks with a non-polynomial activation function can approximate any function* Neural Networks, 861-867, 6, 6, doi:10.1016/S0893-6080(05)80131-5 (1993).
- [51] LU, PU, WANG, HU & WANG *The Expressive Power of Neural Networks: A View from the Width* CoRR arXiv:1709.02540 (2017).
- [52] KIDGER & LYONS *Universal Approximation with Deep Narrow Networks* CoRR arXiv:1905.08539 (2019).
- [53] HERNÁNDEZ & AMIGÓ *Differentiable programming and its applications to dynamical systems* CoRR arXiv:1912.08168 (2020).
- [54] THE ROYAL FAMILY *Texts of speeches given by The Queen and other members of the Royal Family* [Página web de la casa real británica](#) (2020).
- [55] HUGGINGFACE *Transformers v3.4.0* [Documentación de Transformers](#) (2020).
- [56] FACEBOOK's AI RESEARCH LAB (FAIR) *PyTorch v1.6.0* [Documentación de PyTorch](#) (2019).
- [57] GOOGLE BRAIN *TensorFlow v2.0* [Documentación de TensorFlow](#) (2020).
- [58] XIE *Neural Text Generation: A Practical Guide* CoRR arXiv:1711.09534 (2017).
- [59] CHEN, BEEFERMAN & ROSENFIELD *Evaluation Metrics For Language Models* (1998).

- [60] LIN *ROUGE: A Package for Automatic Evaluation of Summaries* Association for Computational Linguistics. Text Summarization Branches Out, 74-81 (2004).
- [61] PAPINENI, ROUKOS, WARD & ZHU *Bleu: a Method for Automatic Evaluation of Machine Translation* Association for Computational Linguistics. Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, 311-318 (2002).
- [62] ÁLVAREZ-MELLADO *A Corpus of Spanish Political Speeches from 1937 to 2019* Proceedings of The 12th Language Resources and Evaluation Conference, European Language Resources Association, 928-932 (2020).
- [63] FACEBOOK's AI RESEARCH LAB (FAIR) *FastText Documentación de FastText* (2018).
- [64] SALLE, IDIART & VILLAVICENCIO *Matrix Factorization using Window Sampling and Negative Samplingfor Improved Word Representations* Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, 419-424, *Documentación de LexVec* (2016).
- [65] MELAMUD, GOLDBERGER & DAGAN *context2vec: Learning Generic Context Embedding with Bidirectional LSTM* Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning, 51-61, *Documentación de Context2Vec* (2016).
- [66] WANG & CHO *BERT has a Mouth, and It Must Speak: BERT as a Markov Random Field Language Model* Proceedings of the Workshop on Methods for Optimizing and Evaluating Neural Language Generation, Association for Computational Linguistics, 30-36, (2019).
- [67] PYTORCH *TextGAN Biblioteca TextGAN* (2020).