TRABAJO FIN DE GRADO

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Análisis de opiniones

Extracción de Entidades y Aspectos con Aprendizaje Profundo para la tarea del Análisis de Opiniones a nivel de Aspecto.

Autor
Nuria Rodríguez Barroso

Directores
Francisco Herrera Triguero
Eugenio Martínez Cámara

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, septiembre de 2018

Agradecimientos

Quiero agradecer a todo aquel que me ha apoyado durante estos 5 años de carrera. Tanto a mis padres por el soporte económico como a mis amigos por el soporte emocional. Gracias a Paco por fijarse en mi desde el principio y a Eugenio por las horas y las ganas depositadas en este trabajo. Por último, pero no menos importante, gracias a Miguel por el apoyo incondicional.

Resumen

En este proyecto se compara el uso de redes de convolución (CNN) y de redes recurrentes basadas en puertas, en concreto la arquitectura Long-Short Term Memory (LSTM) para la extracción de aspectos y entidades, primera fase de un sistema de Análisis de Opiniones a nivel de Aspecto. Por tanto, se desarrollan dos modelos diferentes y se estudiarán las limitaciones y fortalezas de cada uno de ellos. Además, la evaluación y comparación se llevará a cabo en los conjuntos del estado del arte.

A modo introductorio y para ubicar la importancia del problema, se realiza una breve introducción al Análisis de Opiniones. Seguidamente, se introduce el Aprendizaje Automático y más concretamente el Aprendizaje Profundo desde un punto de vista divulgativo.

Los siguientes capítulos se dedican a los fundamentos matemáticos que aunque sencillos, son indispensables para la comprensión de las diferentes redes neuronales que se utilizan. A continuación se introducen teóricamente las redes neuronales de forma general haciendo incapié en las dos estructuras que se van a utilizar: convolucionales y recurrentes basadas en puertas. También se dedica un capítulo a los Campos Condicionales Aleatorios (CRF) y el Algoritmo de Viterbi motivados por el etiquetado de secuencias.

Finalmente, se lleva a cabo el análisis y desarrollo del proyecto de ingeniería propuesto, concluyendo el proyecto con la comparativa de los modelos desarrollados junto con las vías de trabajo futuro.

Palabras clave análisis de opiniones, procesamiento del lenguaje natural, extracción de entidades, redes neuronales, etiquetado de secuencias

Abstract

This work aims to compare two models based on neural network for the task of entity/aspect recognition as part of the task Aspect Based Sentiment Analysis. First of all, we need to contextualize the problem.

Opinion Mininig

The Web 2.0 causes a new (and huge) source of information. This exponential generation of unstructured data was the base of what we know as *Big Data*. Combining this fact with the nature of the human being to give his opinion about everything that surrounds him, gives us an idea of the amount of opinions that we can find on the Internet. This information could be really useful for companies that want to know the opinion of their users. However, the immeasurable amount of information available makes it impossible to hire specialized staff for this task. The necessity of monitoring this task makes the concept of Opinion Mining appear.

As we can imagine, the concept of Opinion Mining covers a huge amount of tasks. During this project, we are going to focus on the entity recognition, the first step of opinion mining at aspect level. The goal of this task is to identify all the entities and classify the polarity of the opinion expressed about them.

Machine Learning

We use machine learning models in order to solve the problem of entity recognition. We define a machine learning algorithm as an algorithm that has the ability to learn. Although we can solve a huge variety of problems with machine learning, the problem that we are going to solve is in the group of classification problems.

The classification problem is the canonical example of supervised machine

learning. This group contains these algorithms that learn from a set with features where each sample has a label associated.

Mathematical Foundation

Deep Learning is considered as a branch of machine learning. These algorithms have a strong mathematical base, so we have to do an introduction to some basic concepts.

The first chapter is dedicated to Probability and Information Theory. First, basic concepts of Probability Theory such as σ-algebra and random variable are introduced. After that, we introduce the concept of probability distribution, developing this theory up to the concept of marginal distribution and conditional probability. Some necessary definitions of Stadistics are introduced such as mean, variance, etc. The foundations of Bayesian Probability are also expound on due to the fact that the majority of deep learning algorithms start from this field of Probability. Finally, a section of this chapter is dedicated to Hidden Markov Models which are essential for the Conditional Random Fields. The second part of this chapter is dedicated to Information Theory, which we superficially address introducing only basic concepts such as crossentropy.

The second chapter is dedicated to Linear Algebra. We start this chapter with some definitions and operators. The main sections in this chapter are dedicated to norms and eigenvalue descomposition, which are essential for deep learning algorithms in order to measure the error and solve linear system of equations respectively.

The next chapter addresses general deep learning algorithms. We introduce the neural network from a theoretical approach. We define the general structure of a neural network. Then, we define the loss function based on crossentropy. After that, we define output units and show some specific examples like sigmoidal units. Finally, we introduce hidden units on the same way that we did with output units.

Once we have a background in general neural network, we focus on the specific architectures that we are going to use in the implementation section. On that approach, we define convolution operator from a mathematical point of view in order to define convolutional neural networks. After this formal intro-

duction, we give an illustrative example of a convolutional neural network. Continuing with the specific architectures, we dedicate a chapter to recurrent neural networks. We start with a motivation in the use of this models in the problem that we have to solve. After that, we define recurrent neural networks and explain the gradient computation. Once we have introduced recurrent neural networks, we describe the problem of the long-term dependencies and propose a solution, the long-short term memory neural networks ($LSTM$). We explain how these models solve the problem of long term dependencies and their effiency. That is why we choose this kind of models to solve our problem.

Finally, we expand on sequence labeling during a whole chapter. We start with the motivation and necessity of this technique due to the nature of our problem. Then, we define Conditional Random Fields in order to find the best sequence of labels for the sample given. As we explain at the end of the section, it requires high performing computaional resources. Therefore, Viterbi Algorithm is introduced. It is a dynamic algorithm that find the best sequence of labels using recursion reducing the amount of computations.

Implementation

Until now, we have focused on the theorical foundation of the deep learning algorithms that we are going to use. From now, we are going to fix our attention in the implementation of these models and the comparison of results.

We start this phase with the analysis of the problem. In this section, we choose the best tools for each task and explain why. After that, we dedicate a chapter to the model design. In this chapter, some diagrams from the UML software engineering framework such as use case diagram or class diagram in order to make visible the structure of the engineering project.

Following the software engineering chapter, we explain the implementation details common to all models. Due to the fact that we are working with phrases in natural language, we have to adjust the data so they can be read by a neural network. On this way, we explain the padding and truncate of phrases and the use of word embedding as a real valued representation of each word. After that, we explain the label system of the problem, composed of four different labels according to IOB annotation scheme and the use of

padding. Finally, we explain the error measures considered to measure the behavior of the models proposed. We have considered three error measures: the macro average, the partial labeling and the chunk labeling depending on what we consider a successful label. The difference between partial labeling and total labeling is to require that the entity is well labeled or just a few words. Regardless of the form of labeling chosen, we consider the precision, the recall and the F1-score.

Finally, we describe the models that we propose. We start with a simple convolutional neural network based on two convolutions with window size of 2 and 3 respectly. After the explanation of the structure, we consider the error measures for this model and try to explain them. For that, we choose some opinions correctly labeled and analyze the proposed labeling. Using this technique, we realize that this model has usual mistakes, for example: the difficulty to find entities formed by several words (more than two) or the difficulty to find entities in opinions composed by several entities (more than two). In this point, we launch the theory that these errors are explained by considering few words for the context that each word knows. In order to fix it, we propose a similar model with bigger window sizes (4 and 6 respectly). Effectively, analyzing the results table we can see that there has been some improvement. We follow the same technique in order to find the limitations of this kind of neural networks. We realize that we solve the problem of finding complex entities in the case of three words, however the system struggles with long entities. Moreover, the error of not finding single-word entities has been introduced and the error of not finding all the entities in sentences with several of them has not been solved. Thus, we conclude that the limitations found in this last model correspond to the limitations of convolutional neural networks to this kind of problems.

The last model we propose is based on a *biLSTM*. This architecture solves the problem of the natural language. We designed a very simple model, based on a biLSTM with 256 hidden units. We observe that this architecture outperforms the models previously presented in most of the evaluation measures. When we analyze specific opinions to find the errors of this model, we see that most common mistakes have disappeared.

In conclusion, we have found a more suitable architecture for this type of problem. Even proposing a very simple structure, we have managed to improve results. This fact opens up doors to many possibilities in this line of

research.

Key words opnion mining, natural language processing (NLP), entity recognition, neural networks, sequence labeling

Yo, **Nuria Rodríguez Barroso**, alumna de la titulación Doble Grado en Ingeniería Informática y Matemáticas en la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 54138901-Y certifico la autoría de este Trabajo de Fin de Grado y autorizo la ubicación de la siguiente copia en la biblioteca del centro para su consulta.

Fdo.: Nuria Rodríguez Barroso

Granada a 7 de septiembre de 2018

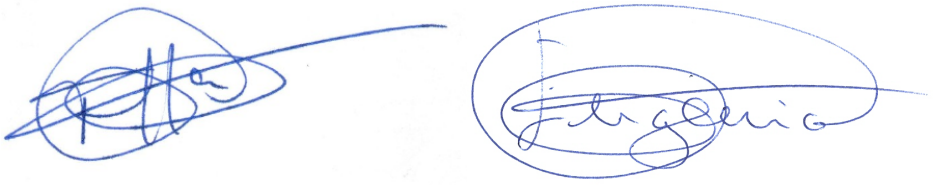
D. **Francisco Herrera Triguero** y D.ª **Eugenio Martínez Cámara**, profesores del Departamento de Ciencias de la Computación e Inteligencia Artificial de la Universidad de Granada.

Informan:

Que el presente trabajo, titulado ***Extracción de Entidades y Aspectos con Aprendizaje Profundo para la tarea del Análisis de Opiniones a nivel de Aspecto***, ha sido realizado bajo su supervisión por **Nuria Rodríguez Barroso**, y autorizamos la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expiden y firman el presente informe en Granada a 7 de septiembre de 2018

Los directores:



Francisco Herrera Triguero **Eugenio Martínez Cámara**

Índice general

1 Introducción **1**
- 1.1 Contexto . . . 1
- 1.2 Motivación . . . 2
- 1.3 Objetivos . . . 5
- 1.4 Requerimientos . . . 6
- 1.5 Planificación y presupuesto . . . 6

2 Análisis de Opiniones **9**
- 2.1 Procesamiento del Lenguaje Natural . . . 9
- 2.2 Concepto de Análisis de opiniones . . . 10
  - 2.2.1 Concepto de Opinión . . . 11
  - 2.2.2 Niveles de clasificación . . . 12

3 Aprendizaje Automático **15**
- 3.1 Concepto de Aprendizaje Automático . . . 15
  - 3.1.1 Tipos de aprendizaje automático. . . . 16
  - 3.1.2 Problema de clasificación . . . 17
- 3.2 Optimización basada en Gradiente Descendente . . . 17
  - 3.2.1 Gradiente Descendente Estocástico . . . 18

4 Probabilidad y Teoría de la Información **23**
- 4.1 Variables Aleatorias . . . 24
- 4.2 Distribuciones de Probabilidad . . . 26
  - 4.2.1 Distribuciones de Probabilidad definidas sobre v.a. discretas . . . 26
  - 4.2.2 Distribuciones de Probabilidad definidas sobre v.a. continuas . . . 27

4.3 Probabilidad Marginal 28
4.4 Probabilidad Condicionada 29
4.5 Conceptos y Resultados básicos de Probabilidad 29
4.6 Probabilidad Bayesiana 31
4.7 Modelos Ocultos de Márkov 32
4.8 Teoría de la Información 33

5 Álgebra Lineal **37**
5.1 Conceptos Básicos 37
5.2 Sistema Lineal de Ecuaciones 40
5.3 Normas 41
5.4 Descomposición en valores propios 43

6 Deep Learning **47**
6.1 Redes Neuronales Prealimentadas 47
6.1.1 Función de coste 51
6.1.2 Unidades de salida 51
Unidades lineales 52
Unidades sigmoidales 52
Unidades con activación softmax 54
6.1.3 Unidades ocultas 55
Unidades lineales rectificadas (ReLu) y generalizaciones 55
Sigmoidal y tangente hiperbólica 56

7 Redes Neuronales Convolucionales **59**
7.1 La operación de Convolución 59
7.1.1 Ejemplo de convolución 61
7.2 *Pooling* 62

8 Modelado de secuencias: Redes Neuronales Recurrentes **63**
8.1 Motivación 63
8.2 Despliegue de Grafos Computacionales 64
8.3 Redes Neuronales Recurrentes 67
8.3.1 Cálculo del Gradiente 69
8.3.2 El problema de las dependencias a largo plazo 71
8.3.3 Redes neuronales recurrentes con memoria a corto-largo plazo 72

9 Etiquetado de secuencias **77**
9.1 Motivación 77
9.2 Modelado de secuencias 78
9.3 Campos condicionales aleatorios de cadena lineal 79
9.4 Algoritmo de Viterbi 82

10 Análisis del problema **89**
10.1 Análisis de requisitos 89
10.1.1 Requisitos funcionales 89
10.1.2 Requisitos no funcionales 90
10.2 Modelo de casos de uso 90
10.2.1 Actores del sistema 91
10.2.2 Diagrama de casos de uso 91
10.3 Modelo de dominio 92
10.4 Herramientas de desarrollo 92

11 Diseño **95**
11.1 Diagrama de clases 95
11.1.1 Patrón de diseño Estrategia 96
11.1.2 Lector de datos 97
11.1.3 Arquitectura software 98

12 Implementación y Experimentación **101**
12.1 Lectura y procesamiento de datos 101
12.1.1 Tokenización 102
12.1.2 Relleno o truncamiento 102
12.1.3 *Word Embeddings* 102
12.2 Características de lo datos utilizados 104
12.3 Etiquetado 104
12.4 Modelos implementados 105
12.5 Medidas de error consideradas 105

13 Modelos y Resultados **109**
13.1 Arquitecturas implementadas 109
13.2 Primer modelo: CNN 110
13.2.1 Estructura 110
13.2.2 Medidas del error 112
13.2.3 Análisis 113

13.3 Segundo modelo: CNN-4-6 115
13.3.1 Estructura 115
13.3.2 Medidas de error 115
13.3.3 Análisis 116
13.4 Tercer modelo: biLSTM 118
13.4.1 Estructura 118
13.4.2 Medidas de error 119
13.4.3 Análisis 121
13.5 Comparativa de modelos 122

14 Conclusiones y trabajo futuro **125**

INTRODUCCIÓN Y MOTIVACIÓN

En esta sección de la memoria desarrollaremos una introducción al Procesamiento del Lenguaje Natural y al Análisis de Opiniones y motivaremos el problema a tratar a lo largo del trabajo: La extracción de entidades y Aspectos.

Para justificar los modelos utilizados para resolver el problema, estudiaremos cuál es el estado del arte en el Aprendizaje Automático para este tipo de problemas, llegando a la conclusión de que los métodos de *Deep Learning* o redes neuronales serán los más adecuados según la literatura.

Capítulo 1

Introducción

1.1 Contexto

La principal característica del ser humano y la que le diferencia del resto de seres vivos es la racionalidad. El ser humano utiliza la razón para la toma de decisiones, contemplando los diferentes escenarios y evaluando la mejor manera de alcanzar sus objetivos. Sin embargo, el ser humano consta de una racionalidad limitada, por lo que el abanico de posibilidades que baraja en cada momento está limitado por su visión de la realidad.

Es por esto que el proceso de toma de decisiones constituye un gran reto. Para facilitar el proceso de toma de decisiones y con el fin de ampliar el abanico de consecuencias contempladas, es muy común la acción de "pedir opinión" a terceros.

La ayuda en la toma de decisiones no es la única función de las opiniones. Las personas también utilizamos las opiniones para expresar nuestro juicio acerca de variados temas. Cuando se produce un intercambio de opiniones ambos participantes se enriquecen con el punto de vista del contrario.

El uso de las opiniones ha evolucionado con el ser humano a lo largo de la historia. Comenzando en los inicios del lenguaje, pasando a estar por escrito con la llegada de la escritura y disparándose con el auge de la difusión de opinión en la prensa con la invención del a imprenta.

Uno de los acontecimientos más influyentes en la sociedad fue la invención de Internet en la segunda mitad del s. XX. Trajo consigo una gran fuente de información de fácil acceso. A principios del s.XXI enfatizó el cambio social que ya estaba ocurriendo un nuevo concepto de Web, la Web 2.0. Este innovador concepto ofrecía a todo usuario de ella la posibilidad de compartir todo tipo de información en Internet. Así fue como se preparó el camino para la llegada de las redes sociales, los blogs, o los foros. Plataformas donde los usuarios podían compartir todo tipo de pensamientos u opiniones sin ningún flitro. También se sumaron al carro de la Web 2.0 las empresas. Muchas de ellas incorporaron la opción de compra *online* suponiendo una nueva fuente de ingresos e incluso surgieron muchas nuevas empresas que llevan a cabo toda su funcionalidad a través de Internet.

1.2 Motivación

[1] En el contexto de la Sección 1.1, tanto el auge de las redes sociales como la incorporación de las empresas al negocio *online* no trajeron consigo solo nuevas fuentes de beneficios. La Web 2.0 supuso una nueva (y enorme) fuente de información. Esta generación exponencial de datos de naturaleza desestructurada en Internet propició el nacimiento de lo que hoy conocemos como *Big Data*.

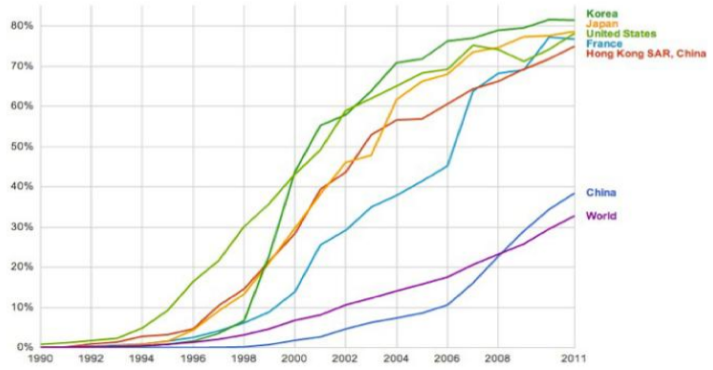


Figura 1.1: Gráfica del crecimiento del *Big Data* en algunos países y en el mundo.

Para hacernos una idea del impacto del nacimiento de la Web 2.0, observamos la Figura 1.1 en la que se representa el crecimiento de *Big Data* en algunos países desarrollados y en el mundo en general. Notamos que en el año 1998 se produce un aumento en el ritmo de crecimiento, coincidiendo con el nacimiento de la Web 2.0. El siguiente cambio de crecimiento más brusco lo encontramos en el año 2006, año de mayor impacto de las redes sociales.

Como ya veníamos advirtiendo, está en la naturaleza del ser humano dar su opinión sobre los temas que le rodean. Esta caracterización del ser humano junto con la cantidad de datos depositados por usuarios en los diferentes portales de Internet nos da una idea de la cantidad de opiniones disponibles. Esta información será de gran utilidad para las empresas que quieran conocer la opinión de los usuarios sobre un determinado producto o, simplemente, para conocer la opinión de la población sobre un determinado tema. Sin embargo, la inmensurable cantidad de información disponible hace que contratar a personal especializado para que se dedique a estudiar las opiniones de un determinado producto o tema de actualidad sería inviable tanto desde el punto de vista económico como desde el temporal. Debido a la necesidad de monitorizar este procedimiento surge el concepto de Análisis de Sentimientos o Minería de Opinión, del que hablaremos en los siguientes capítulos.

Por su estructura de *microblogging* (red social con un número de caracteres limitados para resaltar la opinión), Twitter es un perfecto generador de opiniones para analizar mediante Análisis de Sentimientos. Ya se han llevado a cabo estudios sobre esta plataforma obteniendo muy buenos resultados. Por ejemplo, en las últimas elecciones presidenciales de EEUU se realizó un estudio sobre la opinión de los usuarios de Twitter sobre ambos candidatos. Para este estudio se utilizaron los tweets publicados hasta 43 días antes de las elecciones comenzando el primer día de candidatura. Utilizando expertos para el etiquetado de los datos como positivos o negativos referentes a cada uno de los candidatos y entrenando con un algoritmo de Naïve Bayes consiguieron un 94 % de correlación [1].

El término de Análisis de Sentimientos es un término relativamente reciente. Aunque ya haya tenido éxito en algunos ámbitos, es un campo aún sin explotar pues presenta dificultades que aún no han sido solventadas. Entre estas dificultades se encuentran la subjetividad de las opiniones, las particu-

[1] `https://contently.com/2012/10/24/social-media-sentiment-becomes-factor-in-presidential-campaigns/`

laridades del lenguaje, los contextos, los sarcasmos, entre una interminable lista.

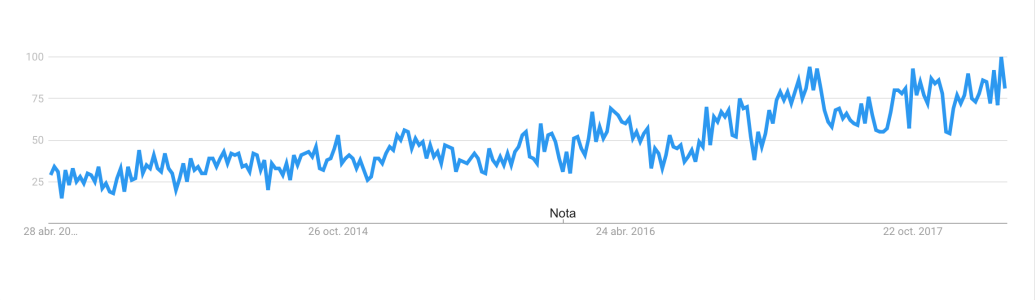


Figura 1.2: Gráfica que representa el interés del análisis de sentimientos en todo el mundo según Google Trends.

Esto hace que se presente como una campo de estudio muy llamativo, con muchas mejoras por hacer y con un futuro muy prometedor. Cada día más personas centran su interés en esta materia (Figura 1.2). De ahí que a lo largo de esta memoria nos dediquemos al estudio de una parte de este amplio campo.

Como ya podemos imaginar, el término Análisis de Sentimientos abarca un gran conjunto de tareas. El trabajo se va a centrar en una de esas tareas en concreto: la extracción de entidades o aspectos, que sería el primer paso del Análisis de Opiniones a Nivel de Aspecto (del ingles *Aspect Based Sentiment Analysis (ABSA*). Esta tarea tiene como fin identificar todas las entidades y sus aspectos y clasificar la opinión que sobre ellos se está expresando en un fragmento de texto.

Para explicar en qué consiste y su importancia, trabajaremos sobre un ejemplo concreto. Imaginemos que queremos analizar la siguiente opinión:

> [1] *El ordenador tiene muy buen procesador, sin embargo la pantalla tiene poca resolución y el precio es muy elevado.*

Si tuviéramos que establecer una polaridad a dichar frase, sería una tarea complicada incluso para una persona. En la frase se nombran tres componentes de un mismo ordenador y se da un opinión sobre cada una de ellas. Así, *buen procesador* tendría connotaciones positivas mientras que *la pantalla tiene poca resolución* y *el precio es muy elevado* negativas. La polaridad ge-

neral del producto dependerá de la importancia que se le dé a cada una de las partes. Como no podemos saber a priori un orden de prioridad establecido, sería muy útil poder dar una polaridad a cada una de las componentes.

Ahora bien, para poder realizar este proceso de forma automatizada nos encontraríamos con dos tareas: en primer lugar la identificación del aspecto y, en segundo lugar, la detección de la polaridad de esta. En este trabajo nos centraremos en la primera de estas tareas.

1.3 Objetivos

El objetivo del proyecto será comparar el uso de redes neuronales de convolución (CNN) y redes neuronales recurrentes basadas en puertas, en concreto la arquitectura Long-Term Short Memory (LSTM) en la tarea de la extracción de aspectos y entidades.

De este modo, se desarrollarán dos modelos neuronales distintos, y se comparará tanto el rendimiento de los mismos como la exactitud en los resultados obtenidos.

Desde un punto de vista más teórico se estudiarán las bases del deep learning, centrándonos en la base matemática de las redes neuronales contempladas. En concreto, nos centraremos en el estudio matemático de las redes neuronales recurrentes pues son de gran interés para nuestro problema.

Así, la estructura del trabajo constaría de los siguientes tomos:

1. Introducción teórica al Análisis de sentimientos.
2. Fundamentos matemáticos del Deep Learning.
3. Implementación del modelo basado en CNN.
4. Implementación del modelo basado en LSTM.
5. Comparativa entre los modelos desarrollados.

1.4 Requerimientos

Los requerimientos del proyecto que desarrollaremos se resumen en:

1. Clasificar/extraer entidades o aspectos mediante una arquitetura basada en CNN.
2. Clasificar/extraer entidades o aspectos mediante una arquitetura basada en LSTM.
3. Comparar los resultados obtenidos por las arquitecturas implementadas.

1.5 Planificación y presupuesto

Para la planificación de tareas en el tiempo se ha realizado un Diagrama de Gantt. Esta herramienta prooprciona un reparto temporal de las tareas de forma muy visual.

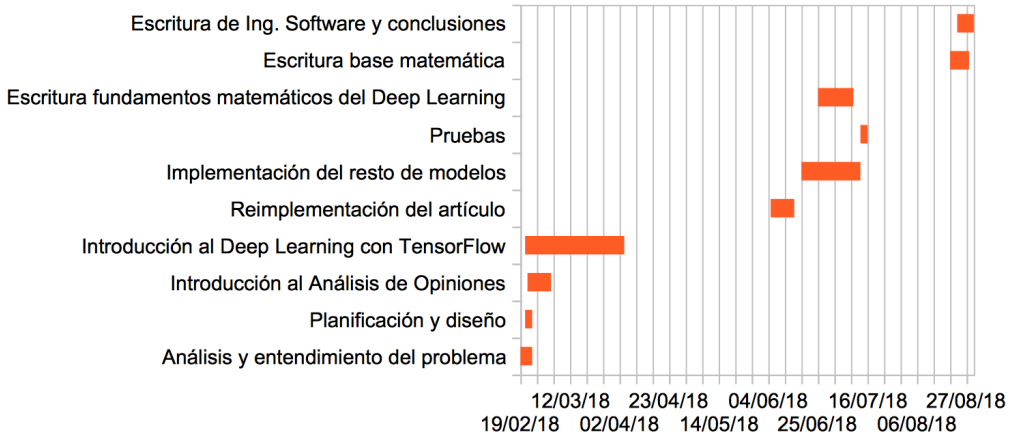


Figura 1.3: Diagrama de Gantt con el reparto de tareas en el tiempo.

Dada la carga lectiva del segundo cuatrimestre del Doble Grado y al mes de vacaciones el reparto de tareas se diferencia en tres grandes bloques como observamos en la Figura 1.3.

Las primeras tareas tienen que ver con el entendimiento del problema e introducción a los algoritmos que se van ha utilizadar. Tras la finalización del curso se realiza la mayor carga de implemetacón y experimentación y se dedican las últimas semanas para el desarrollo del resto de tareas, como escritura de gran parte de la memoria.

En total se han empleado 3 meses y medio en realizar el proyecto.

Para el cálculo del presupuesto se tiene en cuenta:

- **Coste de personal:** dado que el sueldo medio de un Ingenierio Informático Junior en España es de 1800 euros y la cotización social media es de un 32,6 %, el gasto mensual en coste de personal sería de 2670,6 €.

  A partir de los datos anteriores se deduce que el gasto de personal para la realización del proyecto es $3{,}5 \times 2670{,}6 = 9347{,}1$ €.

- **Equipo necesario:** el proyecto se ha realizado en un ordenador personal con las siguientes características:

  1. MacBook Pro 15 pulgadas Pantalla de Retina.
  2. Procesador 2,4 GHz Intel Core i7.
  3. Memoria 8 GB 1600 HMz DDR3.
  4. Gráficos Intel HD Graphics 4000 1536 MB

  Por tanto el gasto en equipo ascendería a los 2000 €.

Por tanto, el presupuesto total del proyecto sería **11347.6** €.

Capítulo 2

Análisis de Opiniones

Para definir el concepto de Análisis de Opiniones es necesaria una breve introducción al Procesamiento del Lenguaje Natural.

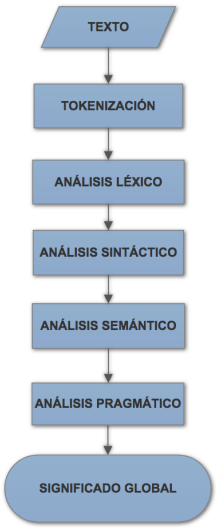
2.1 Procesamiento del Lenguaje Natural

El Procesamiento del Lenguaje Natural (PLN) es un campo de la Inteligencia Artificial que se encarga de estudiar la comunicación que surge entre personas y máquinas. El objeto de estudio del PLN es, por tanto, el lenguaje. Para que las máquinas puedan conseguir capacidad de procesamiento y de generación de lenguaje, es necesario que se tenga comprensión total de la lengua. De esta misión se encarga la Lingüística, que trata de caracterizar y explicar los procedimientos del lenguaje.

El procedimiento usual de un sistema PLN se divide en tres fases principales: análisis sintáctico, semántico y pragmático. En la práctica, estas fases no son suficientes. Esto se debe a que no es viable realizar el análisis sintáctico de un texto sin haber identificado las palabras y las oraciones además de la información morfológica de las palabras que componen la frase. Así, a las tres fases anteriores habría que añadir dos nuevas fases: *tokenización* y análisis léxico.

A continuación observamos el flujo de trabajo que seguiría un sistema PLN. La funcionalidad de la cada una de las fases sería:

1. *Tokenización:* Identificar las unidades en las que se divide el mensaje, *tokens*.
2. Análisis léxico: Obtiene información de los *tokens* a partir de la función que desempeñan en la oración.
3. Análisis sintáctico: Obtiene las relaciones y dependencias existentes entre los elementos que componen la oración.
4. Análisis Semántico: Obtención de la intención del autor a partir de la información obtenida en la fase anterior. Busca descubrir el significado de la frase.
5. Análisis Pragmático: Busca las conexiones entre oraciones para dar un sentido global al discurso.



La búsqueda del entendimiento del lenguaje no es la única misión del PLN. También persigue la generación automática de lenguaje, aunque no va a ser en lo que centremos esta memoria.

2.2 Concepto de Análisis de opiniones

El Análisis de opiniones (AO) recibe muchos nombres dado a la evolución histórica que ha tenido: Análisis de Sentimientos, Minería de Opiniones, etc. A partir de ahora y durante todo el proyecto nos referiremos a él como Análisis de Opiniones.

Definimos el Análisis de Opiniones como el estudio computacional de opiniones, sentimientos y emociones expresadas en un texto. Puede ser considerado como un proceso de clasificación cuyo fin es decidir la polaridad del sentimiento expresada en el texto. La dificultad de esta ciencia recaerá en el objeto de estudio pues se trata de información desestructurada, teniendo que convertirlos en datos estructurados [11].

2.2.1 Concepto de Opinión

Como hemos comentado en la sección anterior, el objetivo del AO consite en la extracción de información a partir de textos. Fundamental para definir ese concepto será la definición de su objeto de estudio, el concepto de opinión.

Una primera aproximación para la definición de opinión la encontramos en el diccionario de la RAE, una opinión es un "juicio o valoración que se forma una persona respecto de algo o de alguien." De esta definición podemos destacar varias cosas. Claramente una opinión es subjetiva, juicio o valor, pero también vemos que se hace énfasis en quién crea la opinión y sobre qué. Empezamos así a vislumbrar que la opinión estará formada por varios elementos.

Para hacer más énfasis en este concepto de opinión formada por varios elementos, veamos un ejemplo concreto:

> *[1] El pasado jueves fui con un grupo de amigos a cenar a un restaurante. [2] Pedí unos macarrones a la carbonara que estaban deliciosos. [3] Sin embargo, el postre no me gustó nada, pedí una tarta de queso. [4] Creo que en este sentido acertó más mi amiga María que pidió la tarta de tres chocolates, ¡le encantó! [5] En cuanto al precio, no estaba nada mal para la cantidad que servían. [6] También hay que tener en cuenta que se encontraba a las afueras, solo se puede acceder en coche.*

Si observamos el texto anterior con la intención de extraer una opinión global sobre el restaurante, encontramos que se da información "contradictoria" pues hay oraciones que expresan opiniones negativas, positivas e incluso neutras. La oración [1] expresa un hecho, sin ningún juicio de valor. Por su parte, las oraciones [2], [4] y[5] expresan sentimiento positivo mientras que las oraciones [3] y [6] expresan alguna opinión negativa. Vemos que cada una de estos juicios se realizan a ciertos elementos del restaurante, por lo que será algo a tener en cuenta dependiendo de si queremos determinar opinión de la entidad (restaurante) o de cada uno de sus componentes. Otro detalle presente en el ejemplo es que no siempre la persona que juzga algo es el autor del texto. En este caso, en la frase [4] el autor expresa que a su amiga le encantó su postre. Destacar también la importancia del tiempo a la hora de realizar un juicio de valor pues nuesta opinión puede variar a lo largo del tiempo.

En este punto, podemos considerar la opinión como una cuádrupla *(o, p, h, t)* donde o representa el objeto sobre el que recae la opinión, p la polaridad

de la misma, h el autor y t el tiempo en el que se produjo.

Si nos fijamos en la frase [3] que expresa un sentimiento negativo, vemos que la opinión negativa no es sobre el restaurante en su conjunto si no que es hacia un postre concreto. Según la definición de opinión como una cuádrupla *(o, p, h, t)* su polaridad recaería sobre el objeto en cuestión, el restaurante. Así, surge la necesidad de considerar que el objeto pueda ser dividido en diferentes componentes. Definimos como entidad e a un producto, servicio, tema, persona, organización o ente. Esta entidad se representa mediante *(T, W)* donde T es una jerarquía de componentes y W el conjunto de atributos de e.

Con esta nueva consideración llegamos a nuestra definición de opinión, introducida en [6]. Una opinión estaría formada por una quíntupla *(e_i, a_{ij}, p_{ijkl}, h_k, t_l)* donde e_i representa la entidad, a_{ij} sería el atributo de e_i sobre el que recae la opinión, p_{ijkl} se correspondería con la polaridad de la opinión que se hace sobre el aspecto a_{ij} en el momento de tiempo t_l por la persona h_k y, por último, h_k y t_l la persona y el tiempo en que se realiza el juicio de valor respectivamente.

2.2.2 Niveles de clasificación

Como ya hemos introducido, el objetivo final del Análisis de Opiniones es la clasificación de un texto según su polaridad. Ahora bien, a la hora de clasificar la polaridad de un texto podemos considerar tres niveles:

1. **Nivel de documento:** Analiza el documento completo y asigna una polaridad global al documento en su conjunto. Es el nivel más general y, por tanto, menos preciso.

2. **Nivel de oración:** Detecta el sentimiento en frases. Está muy relacionado con la subjetividad ya que esta se expresa en frases y no en documentos largos. Obtendríamos una clasificación más precisa del sentimiento.

3. **Nivel de aspectos o entidades:** Detecta el sentimiento u opinión y la entidad a la que se refiere. Es el nivel más preciso de análisis, pues no produce generalizaciones si no que se identifica una polaridad con un aspecto concreto.

Como ya hemos comentado, nuestro proyecto se centrará en el último de estos niveles de clasificación. La importancia de esta especificación queda reflejada en el ejemplo que pusimos en la sección 2.2.1. Además, una vez tengamos clasificada la polaridad de los diferentes aspectos o entidades, es fácilmente extrapolable al resto de niveles de clasificación. Podemos obtener la clasificación a nivel de oración simplemente con la combinación de la polaridad de cada uno de los aspectos que aparecen en la sentencia y análogamente con la clasificación a nivel de texto.

Capítulo 3

Aprendizaje Automático

En este capítulo desarrollaremos los principios básicos de aprendizaje automático para introducir el *Deep Learning* en capítulos posteriores. Motivaremos su uso con algunos ejemplos y nos adentraremos en el problema de clasificación pues es el que desarrollaremos durante el resto de la memoria.

Finalmente hablaremos del gradiente descendente, algoritmo de optimización en el cual se inspirará el Deep Learning.

3.1 Concepto de Aprendizaje Automático

Un algoritmo de **Aprendizaje Automático** es un algoritmo que es capaz de aprender información a partir de ciertos datos. Pero, ¿qué entendemos por aprender? Según [8], "Un programa de ordenador se dice que aprende de una experiencia E con respecto a un conjunto de tareas T y medida de rendimiento P, si su rendimiento como tarea en T, medido con P, mejora tras conocer la experiencia E". Como podemos imaginar, estas tareas y experiencias son muy diversas, de donde surgen gran variedad de técnicas de aprendizaje con estas características.

Desde el punto de vista de las tareas T, el aprendizaje automático es interesante pues nos permite tratar con tareas que son muy difíciles de resolver con programas escritos y diseñados por humanos, ya sea por la complejidad de la solución o por el tamaño de la tarea.

Han sido muchas las aplicaciones de esta clase de algoritmos. En la literatura podemos encontrar varios ejemplos:

- Clasificación
- Clasificación con valores perdidos
- Regresión
- Transcripción
- Detección de anomalías
- Predicción de valores pedidos
- Estimación de densidad

Con respecto a la medida de rendimiento, P, debemos diseñar una medida cuantitativa. La medida por excelencia para casos de clasificación y transcripción es la exactitud o *accuracy* de la solución, definida como la proporción de muestras para las que el modelo ha producido una salida correcta. También puede ser útil en algunos casos la tasa de error o *error rate*, que mide la proporción de muestras para las que el modelo ha producido uan salida incorrecta.

Para aquellas tareas como la estimación de densidad, en las que la salida es continua no tiene sentido utilizar las medidas anteriormente comentadas. En estos casos se utilizarán medidas que proporcionen resultados continuos.

3.1.1 Tipos de aprendizaje automático.

En función de la experiencia E, podemos clasificar los algoritmos de aprendizaje automático en dos grandes grupos: algoritmos supervisados y no supervisados.

- **Aprendizaje Automático no supervisado**: hablamos de este tipo de aprendizaje cuando los datos con los que trabajamos se conforman de un conjunto de datos que representan diferentes características de cada una de las muestras. De este modo, el objetivo suele ser encontrar propiedades importantes para la estructura del conjunto de datos. Un ejemplo clásico de problema de este tipo es el *clustering*, que consiste en la agrupación de los datos en diferentes subconjutnos que contengan características similares.

- **Aprendizaje Automático supervisado**: se obtiene información de un conjunto con características pero donde cada muestra tiene asociada una etiqueta. Así, el objetivo de este tipo de problemas es encontrar una relación entre la etiqueta y las características, pudiendo etiquetar muestras futuras. Un ejemplo de este tipo de problema es la clasificación, en el que entraremos en más detalle a continuación.

3.1.2 Problema de clasificación

El problema de clasificación es el ejemplo más frecuente de aprendizaje supervisado. Consiste en, a partir de una población de entrada, clasificarla en diferentes subpoblaciones o clases. Se conoce previamente el número de clases posibles, por lo que se reduce a decidir elemento a elemento qué clase es la que mejor se ajusta a sus características.

Según el número de clases consideradas podemos distinguir entre:

- **Clasificación binaria:** cuando la clasificación se reduce a dos etiquetas. Se puede ver como un problema de pertenencia o no pertenencia a un subgrupo de la población.
- **Clasificación multiclase:** cuando el conjunto de posibles etiquetas es mayor que dos. En este caso habrá que elegir de entre todas las etiquetas la que mejor se ajute a las características de los elementos de la población a analizar.

El problema a tratar en nuestro caso se trata de un **problema de clasificación multiclase**. En concreto, se tratará de un problema con cuatro etiquetas como se detallará en la Sección 11.

3.2 Optimización basada en Gradiente Descendente

La mayoría de los algoritmos de Deep Learning involucran algún tipo de optimización. La función que queremos optimizar $f(x)$ se denomina **función objetivo**.

Suponemos que tenemos una función $y = f(x)$ con $x, y \in \mathbb{R}$. La derivada de esta función, $f'(x)$ nos proporciona el crecimiento/decrecimiento de la misma. Es decir, nos indica cómo escalar un pequeño cambio en la entrada para obtener el correspondiente cambio en la salida. Esto es, usando la definición clásica de derivada con valores muy pequeños de ϵ:

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon) - f(x)}{\epsilon} \Rightarrow f(x+\epsilon) \approx f(x) + \epsilon f'(x) \tag{3.1}$$

De este modo podríamos minimizar la funciñon f pues sabríamos cómo variar x para realizar una mejora en y. Por ejemplo, si $f(x - \epsilon \text{sign}(f'(x))) < f(x)$ para $\epsilon > 0$ arbitrario, entonces podemos reducir $f(x)$ sin más que mover x en el sentido contrario al signo de la derivada. Esta técnica es lo que se conoce como gradiente descendente para una variable.

En el caso de varias variables, el procedimiento es muy similar. Consideraremos el gradiente de la función $f : \mathbb{R}^{\kappa} \to \mathbb{R}$ en x, $\nabla_x f(x)$ y nos movemos una cantidad $\epsilon > 0$ en misma dirección y sentido contrario de la forma:

$$x' = x - \epsilon \nabla_x f(x) \tag{3.2}$$

El algoritmo multivariante termina cuando $\nabla_x f(x)$ es 0 o muy cercano según una tolerancia prefijada.

Destacar que la velocidad con la que avance el método variará en función del valor tomado para ϵ en ambos casos. Así, si elegimos un valor de ϵ muy pequeño, el método puede avanzar de forma demasiado lenta mientras que si elegimos un valor muy alto puede producirse un efecto *zigzag* alrededor de un óptimo local

3.2.1 Gradiente Descendente Estocástico

Un problema del Aprendizaje Automático radica en que para obtener suficiente generalización, el conjunto de datos sobre el que entrenamos los datos debe ser lo suficientemente grande. Esto conlleva una carga muy elevada de cómputo, pues normalmente la función de coste se calcula como la suma del valor en cada uno de los valores de la muestra.

El Gradiente Descendente Estocástico (GDS) se basa en la idea de subsanar este problema utilizando solo un conjunto pequeño de muestras, llamado **minibatch** $\mathbb{B} = \{x^{(1)}, ..., x^{(m)}\}$, donde m representa una pequeña muestra en función con el total del conjunto de datos.

De esta forma, la estimación del gradiente sería

$$g = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(x^{(i)}, y^{(i)}, \theta)$$

donde $x \in \mathbb{B}$. A continuación, mediante el algoritmo del Gradiente Estocástico descendente se estimaría el gradiente de la siguiente forma

$$\theta \leftarrow \theta - \epsilon g$$

donde ϵ es la tasa de aprendizaje.

El surgimiento del *Deep Learning* se ve motivado por el interés en mejorar al Aprendizaje Automático en ciertos problemas centrales de la Ingeligencia Artificial, como por ejemplo el recoocimiento de imágenes o el procesamiento del lenguaje natural.

FUNDAMENTOS TEÓRICOS

En esta sección de la memoria justificaremos de forma teórica el funcionamiento de los algoritmos de aprendizaje empleados en el resto de la memoria. En una primera parte, de objetivo general y carácter más matemático, abarcaremos conceptos básicos de inferencia estadística, optimización y álgebra que serán necesarios para el entendimiento del resto de la sección. En una segunda parte, de objetivo más concreto, introduciremos los modelos de aprendizaje utilizados al mismo tiempo que justificamos su funcionamiento de forma teórica.

Capítulo 4

Probabilidad y Teoría de la Información

En este capítulo desarrollaremos la Teoría de la Probabilidad y la Teoría de la Información de una forma muy básica para facilitar el entendimiento del resto del proyecto [4].

El Aprendizaje Automático trata con cantidades inciertas y a veces no deterministas (estocásticas). La incertidumbre puede provenir de diversas fuentes: la inherente estocasticidad en el sistema que estamos modelando, de una observación incompleta o incluso de un modelo incompleto.

En el ámbito de la Inteligencia Artificial, utilizamos conceptos y resultados de la Teoría de la Probabilidad con dos propósitos principales. En primer lugar, las leyes de la probabilidad rigen cómo razonan los algoritmos de Inteligencia Artificial, de este modo diseñamos los algoritmos que aproximan las expresiones deseadas usando Teoría de la Probabilidad. En segundo lugar, podemos usar resultados de Probabilidad y Estadística para analizar el comportamiento teórico de algoritmos de este tipo.

Mientras que la importancia de la Teoría de la Probabilidad radica en el estudio de comportamientos inciertos, la Teoría de la Información nos permite cuantificar la cantidad de incertidumbre de una distribución de probabilidad.

4.1 Variables Aleatorias

Para la definición de variable aleatoria (v.a.) necesitamos introducir conceptos básicos de teoría de la medida [14].

La primera definición básica para la teoría de la probabilidad es el concepto de σ-álgebra pues será fundamental para construir el resto de conceptos de esta teoría.

Definition 4.1. *Una familia de subconjuntos de X, representada por Σ, es una σ-**álgebra** sobre X cuando se verifican las siguientes tres propiedades:*

1. *El conjunto vacío $\emptyset$ pertenece a la familia de subconjuntos Σ: $\emptyset \in \Sigma$.*
2. *Si A pertenece a la familia de subconjuntos Σ, entonces su complementario también pertenece: $A \in \Sigma \Rightarrow \overline{A} = X \backslash A \in \Sigma$.*
3. *Si $A_1, A_2, A_3, ...$ es una sucesión de elementos de Σ, entonces la **unión numerable** de todos los conjuntos de la sucesión también es un elemento de Σ: $A_1, A_2, A_3, ... \in \Sigma \Rightarrow \bigcup A_i \in \Sigma$.*

A partir del concepto anterior y con el concepto básico de función de medida de Teoría de la Medida podemos definir el espacio muestral, que contempla todos los posibles resultados de un experimento aleatorio.

Definition 4.2. *Un **espacio muestral** se define como una tripleta $(\Omega, \mathcal{A}, \mu)$ donde:*

1. *Ω es el conjunto de los sucesos elementales.*
2. *$\mathcal{A} \subset \mathcal{P}(\Omega)$ es una colección de subconjuntos de Ω que forma una σ-álgebra de subconjuntos.*
3. *μ es una función de medida de conjuntos que permite asignar probabilidades a los sucesos del espacio muestral.*

El concepto de espacio muestral es fundamental en los espacios probabilísticos, pues parte de su definición a la cual incorpora un conjunto de sucesos de

interés a partir del cual se define la función de probabilidad. El concepto de espacio de probabilidad o espacio probabilístico será crucial en el desarrollo de este capítulo pues a partir de él se modelizan los experimentos aleatorios.

Definition 4.3. *Un* ***espacio de probabilidad*** *se define como la tripleta* $(\Omega, \mathcal{B}, P)$ *donde:*

1. Ω *es el espacio muestral (sucesos elementales).*
2. $\mathcal{B}$ *es una colección de sucesos aleatorios que forma una* σ*-álgebra sobre* Ω*.*
3. P *es una función de probabilidad que asigna una probabilidad a cada uno de los sucesos.*

Para entender la definición anterior necesitamos el concepto de función de probabilidad, que no es más que una extensión del concepto de función de medida verificando los **Axiomas de Kolgomorov**:

Definition 4.4. *Decimos que una función de medida* P *definida sobre una* σ*-álgebra* $\mathcal{B}$ *es una* ***medida o función de probabilidad*** *si verifica que:*

1. ***Axioma 1****: La función toma valores en el intervalo cerrado* $[0, 1]$

$$0 \leq \mathcal{P}(A) \leq 1 \qquad \forall A \in \mathcal{B}$$

2. ***Axioma 2****: La probabilidad del total es 1 y la del elemento vacío es 0:*

$$P(\mathcal{B}) = 1 \qquad P(\emptyset) = 0$$

3. ***Axioma 3****: Si* $A_1, A_2, A_3, ...$ *son sucesos de* $\mathcal{B}$ *disjuntos dos a dos entonces la probabilidad de la unión es la suma de las probabilidades:*

$$P(A_1 \cup A_2 \cup A_3 \cup ...) = \sum P(A_i)$$

En este punto contamos con las herramientas necesarias para introducir el concepto de variable aleatoria:

Definition 4.5. *Una **variable aleatoria (v.a.)** X es una función real definida en el espacio de probabilidad $(\Omega, \mathcal{B}, P)$ asociada a un determinado experimento aleatorio, esto es:*

$$X : \Omega \to \mathbb{R}$$

Con respecto al número de diferentes valores que puede tomar, una variable aleatoria puede ser:

1. **Discreta**: toma valores en un conjunto finito.
2. **Continua**: puede tomar un conjunto inifinito de valores.

4.2 Distribuciones de Probabilidad

Una **distribución de probabilidad** de una variable aleatoria es una función que asigna a cada valor posible de la v.a. (suceso) una probabilidad ($[0, 1]$) de que este ocurra.

En función del tipo de v.a. sobre el cual definamos la distribución de probabilidad podemos diferenciar dos tipos de distribuciones de probabilidad:

4.2.1 Distribuciones de Probabilidad definidas sobre v.a. discretas

Las distribuciones de probabilidad definidas sobre v.a. discretas se definen mediante la **función de probabilidad** o **función masa de probabilidad**. Esta función asocia a cada punto del espacio muestral (suceso) una probabilidad de ocurrencia.

Definition 4.6. *La **función masa de probabilidad P** definida sobre el espacio muestral $\mathcal{A}$ de la variable aleatoria X asigna a cada punto $x_i \in \mathcal{A}$ una probabilidad de que este ocurra. Esto es,*

$$P(x_i) = p_i$$

donde p_i es la probabilidad del suceso $X = x_i$.

La función masa de probabilidad verifica las siguientes propiedades:

Proposition 1. *Sea P una función de masa de probabilidad definida sobre una variable aleatoria discreta X, entonces:*

1. *El dominio de P serán todos los posibles valores de X.*
2. *$0 \leq P(x) \leq 1 \quad \forall x \in X$. De esta forma, una probabilidad de 0 denotará un suceso imposible y una probabilidad de 1 un suceso certero.*
3. *$\sum_{x \in X} P(x) = 1$. Esta propiedad se conoce como **normalización** y es crucial pues en caso de incumlpirse podríamos obtener sucesos con probabilidades mayores que 1.*

4.2.2 Distribuciones de Probabilidad definidas sobre v.a. continuas

En el caso de las distribuciones de probabilidad definidas sobre v.a. continuas utilizamos la **función de densidad de probabilidad** para describirlas.

Definition 4.7. *Una función p es una función de densidad de probabilidad sobre la v.a. continua X si satisface:*

1. *El dominio de p serán todos los posibles valores de X.*
2. *$p(x) \leq 0 \quad \forall x \in X$.*
3. *$\int p(x)dx = 1$*

A diferencia de la función masa de probabilidad, esta función no devuelve la probabilidad para un determinado punto del espacio muestra si no que devuelve la posibilidad de pertenecer a una región infinitesimal de volumen dx.

Proposition 2. *Para conocer la función masa de probabilidad de que x pertenezca a un intervalo concreto, bastaría con integrar la función de densidad de probabilidad en dicho intervalo.*

Por ejemplo, con la notación anterior en el caso univariante tendríamos:

$$P(x \in [a,b]) = P(a \leq x \leq b) = \int_a^b p(x)dx = \int_{[a,b]} p(x)dx$$

4.3 Probabilidad Marginal

En ocasiones, conocemos la distribución de probabilidad de una distribución sobre un conjunto de variables pero nos interesa conocer la distribución sobre un subconjunto de ellas. La distribución resultante sería conocida como **distribución marginal de probabilidad** del subconjunto de variables elegidas.

Obtención de las probabilidades marginales:

1. **Caso discreto** : Consideramos las v.a. discretas X e Y y su función de probabilidad conjunta $P(X,Y)$. Para obtener la distribución marginal de cualquiera de las v.a. bastaría con considerar la regla de la suma para la otra variable. Esto es:

$$\forall x \in X, \;\; P(X = x) = \sum_y P(X = x, Y = y)$$

2. **Caso continuo:** Para v.a. continuas X e Y, conocida la función de densidad de probabilidad $p(X,Y)$. Para conocer la marginal con respecto a una de sus variables, usamos la integración en la otra variable. De esta forma:

$$p(X) = \int p(X,Y)dY$$

4.4 Probabilidad Condicionada

En muchos casos, estaremos interesados en conocer la probabilidad de ocurrencia de un suceso en el caso de que otro suceso haya ocurrido. A este tipo de probabilidades se les llama **probabilidades condicionadas**. Notamos a la probabilidad de que $X = x$ dado que $Y = y$ como $P(X = x \mid Y = y)$ se calcula mediante la siguiente fórmula:

$$P(X = x \mid Y = y) = \frac{P(X = x, Y = y)}{P(Y = y)} \tag{4.1}$$

4.5 Conceptos y Resultados básicos de Probabilidad

Un resultado muy básico de la Teoría de la Probabilidad pero que presenta múltiples utilidades prácticas es la **Regla de la Cadena:**

Proposition 3. *Toda probabilidad conjunta sobre un conjunto de v.a. puede ser descompuesta en distribuciones condicionales sobre una sola variable.*

Demostración. La demostración es inmediata haciendo uso de la definición de probabilidad condicionada. Aplicando la definición de forma sucesiva obtenemos la siguiente descomposición

$$P(X^{(1)}, X^{(2)}, ..., X^{(n)}) = P(X^{(1)}) \prod_{i=2}^{n} P(X^{(i)} \mid X^{(1)}, ..., X^{(i-1)}) \tag{4.2}$$

la cual es una multiplicación de distribuciones condicionales sobre una sola variable como queríamos demostrar.

□

Otro concepto básico es el concepto de independencia entre v.a. Para simplificar la definición, definimos el concepto para dos v.a:

Definition 4.8. *Dos v.a. X e Y son **independientes** si la distribución de probabilidad puede ser expresada como producto de dos factores, uno involucrando solo a X y otro a Y.*

Esto es,

$$p(X = x, Y = y) = p(X = x)p(Y = y) \qquad \forall x \in X, y \in Y \tag{4.3}$$

A partir de esta definición se definiría la independencia en un conjunto como la independencia dos a dos de las v.a. que lo componen.

Son cruciales para el estudio estadístico y probabilístico de la muestra de una v.a. las siguientes medidas:

- **Esperanza matemática:** se define la esperanza matemática de una función $f(X)$ con respecto a una distribución de probabilidad $P(X)$ como el valor medio que toma f cuando X sigue la distribución P. Formalmente:

  Definition 4.9. *Se define la **esperanza matemática** en el caso discreto como*

  $$\mathbb{E}_{X\sim P}[f(x)] = \sum_X P(X)f(X) \tag{4.4}$$

  Definition 4.10. *Se define la **esperanza matemática** en el caso continuo como*

  $$\mathbb{E}_{X\sim P}[f(X)] = \int_X P(X)f(X) \tag{4.5}$$

- **Varianza :** representa una medida de dispersión de los datos. Formalmente:

  Definition 4.11. *Se define la **varianza** de una v.a. como*

  $$Var(f(X)) = \mathbb{E}\left[(f(X) - \mathbb{E}[f(X)])^2\right] \tag{4.6}$$

- **Covarianza:** proporciona una aproximación de cómo de relacionadas linealmente están dos variables.

  Definition 4.12. *Se define la* ***covarianza*** *entre una función $f(X)$ y una función $g(Y)$ con respecto a una distribución de probabilidad P como*

$$Cov(f(X), g(Y)) = \mathbb{E}\left[(f(X) - \mathbb{E}[f(X)])(g(Y) - \mathbb{E}[g(Y)])\right] \tag{4.7}$$

  A partir de esta definición definimos la **matriz de covarianzas** de un vector aleatorio $X \in \mathbb{R}^n$ como una matriz $n \times n$ donde

$$Cov(\mathbf{X})_{i,j} = Cov(X_i, X_j)$$

4.6 Probabilidad Bayesiana

En ocasiones, dadas dos variables aleatorias X e Y, nos vemos en la tesitura de conocer $P(Y \mid X)$ y necesitar conocer $P(X \mid Y)$.

Todo el estudio de las probabilidades a posterori se conoce como el estudio de la Probabilidad Bayesiana Objetiva, rama de la probabilidad en la que se basan la mayoría de los procedimientos de Aprendizaje Automático [2].

Dado el propósito de este trabajo nos centraremos en la **Regla de Bayes**, qu es el principio del desarrollo de toda la teoría y la cual nos permite calcular la probabilidad a posteriori deseada conociendo $P(X)$ de la siguiente forma:

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{P(Y)} \tag{4.8}$$

Sustituyendo $P(Y)$ (a priori desconcido) por su definición como la marginal de $P(Y \mid X)$ con respecto a X tenemos.

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{\sum_X P(Y \mid X)P(X)} \tag{4.9}$$

en el caso discreto, y

$$P(X \mid Y) = \frac{P(Y \mid X)P(X)}{\int_X P(Y \mid X)P(X)} \tag{4.10}$$

en el caso continuo.

4.7 Modelos Ocultos de Márkov

En esta sección se introducen una serie de conceptos que serán necesarios para el entendimiento de los métodos se utiliza para el etiquetado secuencial.

Definition 4.13. *Se define un* ***proceso estocástico*** *como un conjunto de variables aleatorias que evolucionan en función del tiempo.*

Con el objetivo de definir proceso de Márkov definimos la propiedad de Markov como la carencia de memoria.

Definition 4.14. *Se dice que un conjunto de procesos estocásticos verifican la* ***propiedad de Márkov*** *si la distribución de probabilidad del valor futuro de una variable aleatoria X_{n+1} depende únicamente de su valor presente X_n. Esto es,*

$$P(X_{n+1} \mid X_n, X_{n-1}...., X_1) = f(X_n) \tag{4.11}$$

A los procesos estocásticos que verifican esta propiedad se les conoce como ***procesos de Márkov****.*

Por último, necesario para la introducción de los *CRF* del ingés *Conditional Random Fields*, definimos el concepto **modelo oculto de Márkov** o **HMM** (por sus siglas en ingles, *Hidden Márkov Model*).

Un **modelo oculto de Márkov (HMM)** es un modelo estadístico que modela un proceso de Márkov de parámetros desconocidos.

Definimos formalmente un HMM:

Definition 4.15. *Se denota un* ***HMM*** *mediante quíntupla* (S, V, π, A, B) *donde:*

1. $S = \{1, 2, ..., n\}$ *es el* ***conjunto de estados****. El estado inicial se denota como* s_t*.*
2. $V = \{v_1, v_2, ..., v_m\}$ *es el conjunto de* ***valores observables en cada estado*** *donde* m *es el número de palabras posibles.*
3. $\pi = \{\pi_i\}$ *son las* ***probabilidades inciales*** *donde* pi_i *representa la probabilidad de que el primer estado sea* S_i*.*
4. $A = \{a_{ij}\}$ *es el conjunto de* ***probabilidades de transición entre estados*** *donde* a_{ij} *es la probabilidad de estar en el estado* j *en el instante* t *habiendo estado en* i *en el instante* $t-1$*. Esto es*

$$a_{ij} = P(s_t = j \mid s_{t-1} = i) \tag{4.12}$$

5. $B = \{b_j(v_k)\}$ *es el conjunto de* ***probabilidades de las observaciones*** *donde* $b_j(v_k)$ *representa la probabilidad de observar* v_k *cuando se está en el estado* j *en el instante de tiempo* t*. Esto es*

$$b_j(v_k) = P(o_t = v_k \mid s_t = j) \tag{4.13}$$

La ***secuencia de estados observables*** *se denota como* $O = \{ o_1, o_2, ..., o_T, \}$*.*

4.8 Teoría de la Información

La Teoría de la Información es una rama de las matemáticas aplicadas que se encarga de medir y estudiar la información presente en cualquier intercambio de información o señal. Aunque esta rama del conocimiento es fundamental para muchas aplicaciones de la Ingeniería Informática, en esta memoria nos centramos en su aplicación para caracterizar distribuciones de probabilidad o para cuantificar la similitud entre distribuciones.

Definition 4.16. *Se define el concepto de* ***autoinformación*** *de un suceso o evento $X = x$ bajo una cierta distribución de probabilidad P como*

$$I(x) = -\log P(x) \tag{4.14}$$

La definición anterior contempla tratar solo con un resultado. Para ello introducimos la **entropía de Shannon** como

Definition 4.17. *Definimos la* ***entropía de Shannon*** *para medir la verosimilitud en toda la distribución de probabilidad como*

$$H(X) = \mathbb{E}_{X\sim P}[I(X)] = -\mathbb{E}_{X\sim P}[\log P(X)] = H(P) \tag{4.15}$$

Cuando la distribución X es continua, la entroía de Shannon es conocida como ***entropía diferencial****.*

En el caso de que tengamos dos distribuciones de probabilidad diferentes $P(X)$ y $Q(X)$ sobre la misma v.a. X, podemos medir la verosimilitud entre dichas distribuciones con la divergencia de Kullback-Leibler.

Definition 4.18. *En el ámbito anterior, definimos la* ***divergencia de Kullback-Leibler (divergencia KL)*** *como*

$$D_{KL}(P \,||\, Q) = \mathbb{E}_{X\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{X\sim P}\left[\log P(x) - \log Q(x)\right] \tag{4.16}$$

Una simplificación de la medida anterior es la siguiente medida.

Definition 4.19. *En el ámbito anterior, se define la* ***entropía cruzada*** *como*

$$H(P, Q) = -\mathbb{E}_{X\sim P} \log Q(X) \tag{4.17}$$

La importancia de esta medida radica en que al minimizar la entropía cuzada con respecto a Q se minimiza la divergencia KL, dado que Q no aparece en el término omitido en la simplificación. Por este motivo, esta medida será la que utilizaremos en la práctica cuando queramos minimizar la incertidumbre entre dos distribuciones de probabilidad sobre la misma v.a.

Capítulo 5

Álgebra Lineal

[4] En este capítulo introduciremos los conceptos básicos de álgebra lineal. El buen entendimiento de los procedimientos de álgebra lineal es esencial para el entendimiento de muchos algoritmos de Aprendizaje Automático y, sobretodo, de Aprendiaje Profundo.

5.1 Conceptos Básicos

El estudio del álgebra lineal involucra diferentes tipos de objetos matemáticos:

- **Escalares:** Un escalar se correspondería con un número, a diferencia del resto de estructuras que introduciremos a continuación.
- **Vectores:** Un vector es un conjunto ordenado de números. Podemos identificar cada elemento individual por su índice en el vector.
- **Matrices:** Una matriz es un conjunto ordenador de números de dos dimensiones. De esta forma, cada elemento estará identificado por dos índices.
- **Tensores:** En ocasiones, necesitaremos conjuntos de más de dos ejes. Surgen de esta forma los tensores, como números organizados en una cuadrícula regular con un número variable de ejes.

Una operación importante de las matrices es la **transposición**. Esta operación equivale a obtener la imagen de un espejo a través de la diagonal. Formalmente:

Definition 5.1. *Dada una matriz* $\boldsymbol{A}$*, definimos su* ***transpuesta*** *la cual notamos como* $\boldsymbol{A}^T$ *verificando*

$$(A^T)_{i,j} = A_{j,i} \tag{5.1}$$

Aunque la transposición sea una operación propia de las matrices, dado que podemos considerar a los vectores como matrices fila y a los escalares como una matriz de una sola entrada, este concepto es fácilmente aplicable a ambas estructuras.

Notar que el transpuesto de un escalar, es siempre el mismo: $a = a^T$.

En cuanto a las **operaciones** con estas estructuras:

1. Podemos sumar matrices siempre que estas tengan la misma dimensión de la siguiente forma

$$\mathbf{C} = \mathbf{A} + \mathbf{B} \Rightarrow C_{i,j} = A_{i,j} + B_{i,j} \tag{5.2}$$

2. Podemos sumar o multiplicar un escalar a una matriz simplemente actuando elemento a elemento

$$\mathbf{C} = a\mathbf{A} + c \Rightarrow C_{i,j} = aA_{i,j} + c \tag{5.3}$$

3. **Producto de matrices:** El producto matricial es una de las operaciones del álgebra lineal que más se utilizan en los algoritmos de Aprendizaje Automático.

   Definition 5.2. *Sea* $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ *y* $\boldsymbol{B} \in \mathbb{R}^{m\times k}$ *entonces, el* ***producto matricial*** *de* $\boldsymbol{A}$ *y* $\boldsymbol{B}$ *es una nueva matriz* $\boldsymbol{C} = \boldsymbol{AB} \in \mathbb{R}^{n\times k}$ *verificando*

$$C_{i,j} = \sum_{k=1}^{m} A_{i,k}B_{k,j} \tag{5.4}$$

Propiedades del producto matricial:

(a) El producto matricial es distributivo

$$\mathbf{A}(\mathbf{B}+\mathbf{C}) = \mathbf{AB}+\mathbf{AC} \tag{5.5}$$

(b) El producto matricial es asociativo

$$(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC}) \tag{5.6}$$

(c) El producto matricial no es conmutativo

$$\mathbf{AB} \neq \mathbf{BA} \tag{5.7}$$

(d) La transpuesta de un producto es el producto inverso de sus transpuestas

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T \tag{5.8}$$

Definimos a continuación una herramienta muy poderosa para la resolución de ecuaciones lineales, la inversión de matrices. Para ello necesitamos introducir en primer lugar el concepto de matriz identidad.

Definition 5.3. *Definimos la* ***matriz identidad*** *como aquella que no modifica ningún vector cuando lo multiplicamos por dicha matriz. Denotamos a la matriz identidad de dimensión n como* $\boldsymbol{I}_n \in \mathbb{R}^{n\times n}$ *y dicha matriz verifica*

$$\forall \boldsymbol{x} \in \mathbb{R}^n, \boldsymbol{I}_n\boldsymbol{x} = \boldsymbol{x} \tag{5.9}$$

Dicha matriz se compone de elementos 1 en la diagonal y 0 en el resto de posiciones.

Definition 5.4. *La* ***matriz inversa*** *de* $\boldsymbol{A}$ *se denota como* $\boldsymbol{A}^{-1}$ *y es la matriz que verifica*

$$\boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{I}_n \tag{5.10}$$

Un último concepto que introducimos en este apartado es el **determinante**. Es un concepto muy sencillo pero a la vez útil para conocer propiedades de dicha matriz.

Definition 5.5. *Se define el* ***determinante*** *de una matriz cuadrada* $\boldsymbol{A}$ *y se denota por* $det(\boldsymbol{A})$ *a la suma de los* $n!$ *productos formados por los* $n-$*factores que se obtienen de multiplicar* n*-elementos de la matriz de tal forma que cada producto contenga un solo elemento de cada fila y de cada columna de* $\boldsymbol{A}$*.*

Aunque la definición parezca algo complicado, en realidad es un concepto muy sencillo. Más adelante veremos una caracterización que facilita su cálculo.

5.2 Sistema Lineal de Ecuaciones

En este punto podemos introducir un importante concepto del álgebra lineal, los sistemas lineales de ecuaciones.

Definimos un sistema lineal de ecuaciones mediante la ecuación

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{5.11}$$

donde $\mathbf{A} \in \mathbb{R}^{m \times n}$ es una matriz conocida, $b \in \mathbb{R}^m$ es un vector conocido y $x \in \mathbb{R}^n$ es un vector de variables que queremos conocer. Cada elemento x_i de $\mathbf{x}$ es una de las variables desconocidas.

Cada fila de $\mathbf{A}$ y cada elemento de $\mathbf{b}$ generan una nueva restricción.

$$\mathbf{A}_{1,:}\mathbf{x} = \mathbf{b}_1 \tag{5.12}$$

$$\mathbf{A}_{2,:}\mathbf{x} = \mathbf{b}_2 \tag{5.13}$$

$$\mathbf{A}_{3,:}\mathbf{x} = \mathbf{b}_3 \tag{5.14}$$

...

$$\mathbf{A}_{m,:}\mathbf{x} = \mathbf{b}_m \tag{5.15}$$

o de forma más explícita

$$\mathbf{A}_{1,1}x_1 + \mathbf{A}_{1,2}x_2 + ... + \mathbf{A}_{1,n}x_n = \mathbf{b}_1 \tag{5.16}$$

$$\mathbf{A}_{2,1}x_1 + \mathbf{A}_{2,2}x_2 + ... + \mathbf{A}_{2,n}x_n = \mathbf{b}_2 \tag{5.17}$$

$$\mathbf{A}_{3,1}x_1 + \mathbf{A}_{3,2}x_2 + ... + \mathbf{A}_{3,n}x_n = \mathbf{b}_3 \tag{5.18}$$

$$...$$

$$\mathbf{A}_{m,1}x_1 + \mathbf{A}_{m,2}x_2 + ... + \mathbf{A}_{m,n}x_n = \mathbf{b}_m \tag{5.19}$$

La mayoría de las operaciones en el Aprendizaje Profundo (redes neuronales) se basan en este tipo de ecuaciones incorporando operaciones de tipo no lineal.

5.3 Normas

Otro concepto del álgebra lineal muy utilizado en el Aprendizaje Profundo son las normas de vectores y matrices. Se utilizan diferentes tipos de normas para las operaciones de regularización.

Definition 5.6. *Una función $f : \mathbb{R}^n \to \mathbb{R}_0^+$ es una norma si satisface que, $\forall x \in \mathbb{R}^n$:*

1. $f(x) = 0 \Rightarrow x = 0$
2. $\forall y \in \mathbb{R}^n$ *se cumple la* ***desigualdad triangular****, esto es,* $f(x + y) \leq f(x) + f(y)$
3. $\forall \alpha \in \mathbb{R}$, $f(\alpha x) = \alpha f(x)$

La norma más usada es la norma L^p, que se define para $p \in (1, \infty)$

Definition 5.7. *Sea $x \in \mathbb{R}^n$ se define la norma L^p de x que se denota por $|| \ x \ ||_p$ mediante la fórmula*

$$|| \ x \ ||_p = \left(\sum_i | \ x_i \ |^p \right)^{\frac{1}{p}} \tag{5.20}$$

Los casos particulares de esta norma más utilizados son:

1. **Norma euclídea:** que se corresponde con L^2 y representa la distancia euclídea al origen. Es la más utilizada en Aprendizaje Automático y se suele denotar como $|| \ x \ ||$ para simplificar notación. Es común medir el vector utilizando el cuadrado de la norma euclídea dado su fácil cálculo, pues basta con considerar xx^T.

2. **Norma $\mathbf{L^1}$:** también muy utilizada en Aprendizaje Automático y simplificada como

$$|| \ x \ ||_1 = \sum_i | \ x_i \ | \tag{5.21}$$

   Su uso más frecuente es cuando la diferencia entre valores nulos y no nulos es muy importante.

Otra norma muy utilizada en el Aprendizaje Automático es la **norma del máximo** definida mediante la fórmula

$$|| \ x \ ||_\infty = \max_i | \ x_i \ | \tag{5.22}$$

Esta norma se utiliza por su fácil cálculo pues se ahorra grandes sumatorios reduciendo el vector a un simple valor absoluto.

5.4 Descomposición en valores propios

La importancia de la descomposición en valores propios radica en la obtención de información a partir de una descomposición de la matriz. Además, facilita ciertos cálculos como por ejemplo el cálculo de potencias de matrices. Una de los métodos más comunes es la descomposición en valores propios, el cual trataremos en este apartado.

Definition 5.8. *Un **vector propio** de una matriz cuadrada $\boldsymbol{A}$ es un vector $\boldsymbol{v}$ distinto de cero cuya multiplicación por $\boldsymbol{A}$ altera solo la escala de $\boldsymbol{v}$, esto es*

$$\boldsymbol{A}\ \boldsymbol{v}\ = \lambda \boldsymbol{v} \tag{5.23}$$

*El escalar λ es conocido como **valor propio** correspondiente al vector propio $\boldsymbol{v}$.*

Una de las aplicaciones de los valores propios es el cálculo del gradiente.

Proposition 4. *Dada una matriz cuadrada $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, entonces el determinante $det(\boldsymbol{A})$ se corresponde con el producto de sus valores propios.*

$$det(\boldsymbol{A}) = \prod_i \lambda_i \tag{5.24}$$

Para entender la importancia de este tipo de vectores, veámos la descomposición matricial que producen.

Para ello, supongamos que una matriz $\mathbf{A}$ tiene n vectores propios linealmente independientes, $\{v^{(1)}, v^{(2)}, ..., v^{(n)}\}$ con sus correspondientes valores propios $\{\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(n)}\}$. Podemos concatenar todos los vectores propios formando una matriz $\mathbf{V}$ con un vector propio por columna: $\mathbf{V} = [v^{(1)}, v^{(2)}, ..., v^{(n)}]$. De la misma forma, podemos concatenar los valores propios creando el vector $\lambda = [\lambda^{(1)}, \lambda^{(2)}, ..., \lambda^{(n)}]$.

La **descomposición en valores propios** de $\mathbf{A}$ estaría dada por

$$\mathbf{A} = \mathbf{V} diag(\lambda) \mathbf{V}^T \tag{5.25}$$

El procedimiento para la descomposición de matrices cuadradas dados sus vectores y valores propios es sencillo. El problema lo crea la búsqueda de estos vectores y valores.

El cálculo de vectores y valores propios no siempre es sencillo y no siempre es posible con valores reales. Afortunadamente, en la mayoría de las aplicaciones en el Aprendizaje Automático las matrices con las que tratamos son simétricas (representan distancias) por lo que podemos aplicar el siguiente resultado que nos asegura la existencia de tal descomposición en valores reales.

Proposition 5. *Toda matriz* $\boldsymbol{A}$ *simétrica puede descomponerse en valores propios reales de la forma*

$$\boldsymbol{A} = \boldsymbol{Q} \Lambda \boldsymbol{Q}^T \tag{5.26}$$

donde $\boldsymbol{Q}$ *es una matriz ortogonal compuesta de los vectores propios de* $\boldsymbol{A}$ *y* Λ *es una matriz diagonal.*

A partir de los valores propios podemos realizar una simple clasificación de las matrices cuadradas:

1. **Matrices Definidas Positivas:** son aquellas cuyos valores propios son todos mayores que 0.
2. **Matrices Semidefinidas Positivas:** son aquellas cuyos valores propios son todos mayores o iguales que 0.
3. **Matrices Definidas Negativas:** son aquellas cuyos valores propios son todos menores que 0.
4. **Matrices Semidefinidas Negativas:** son aquellas cuyos valores propios son todos menores o iguales que 0.

Esta clasificación es interesante pues si un matriz $\mathbf{A}$ es semidefinida positiva se garantiza que $\forall x, x^T A x \geq 0$, lo cual es un requisito en ciertos algoritmos de Aprendizaje Automático.

El principal problema de esta descomposición es la exigencia de que la matriz sea cuadrada para el desarrollo de toda la teoría. Existen otros métodos

que no precisan de esta exigencia como la *descomposición en valores singulares*. Sin embargo, para nuetro propósito en este trabajo es suficiente con la descomposición de matrices cuadradas pues son las más utilizadas en los algoritmos de Aprendizaje Automático.

Capítulo 6

Deep Learning

En este capítulo nos centramos en las técnicas implementadas a lo largo de la memoria, técnicas de *Deep Learning*. Los diferentes modelos que implementaremos se basan en redes neuronales, por lo que dedicaremos la primera sección de este capítulo a hablar sobre redes neuronales prealimentadas. También se tratarán las características generales de las redes neuronales que nos servirán como base introductoria de los modelos desarrollados.

A continuación, desarrollaremos en profundidad los modelos implementados: redes neuronales convolucionales y redes neuronales recursivas (LSTM) [4].

6.1 Redes Neuronales Prealimentadas

Las **redes neuronales** son el ejemplo más típico de modelos de *Deep Learning*. El objetivo de esta clase de algoritmos es aproximar una función f^*. Por ejemplo, para un clasificador $y = f^*(x)$ que asocia a cada entrada x una categoría y. Así, una red neuronal prealimentada define una asociación $y = f(x; \theta)$ y aprende los valores de los parámetros θ que ofrecen la mejor aproximación de la función.

Esta clase de modelos se llaman **prealimentados** porque la información fluye a través de la función siendo evaluados desde x, mediante los cálculos intermedios usados para definir f y finalmente hasta la salida y. Por tanto, nunca se forman ciclos entre las conexiones de las unidades, la información se

evalúa siempre hacia delante a través de las diferentes capas intermedias. No existe ningún tipo de retroalimentación en las que la salida de alguna capa de la red vuelva a ser entrada del modelo.

Las redes neuronales son llamadas **redes** pues se definen mediante la composición de diferentes funciones. Este modelo de sucesivas composiciones se puede ver como un grafo dirigido acícilo.

El número de capas que tenga la red definen la **profundidad** del modelo. Las capas se van nombrando de forma sucesiva siendo la primera la primera capa o *capa de entrada*, *segunda capa*, y así sucesivamente hasta la capa final denominada *capa de salida*. El objetivo del modelo es conducir la función de entrada $f(x)$ para que se ajuste a $f^*(x)$.

Este tipo de algoritmos están inspirados en la neurociencia, en concreto en las neuronas de los seres vivos. Cada capa del modelo está formado por un número de unidades. El número de unidades en cada capa define la **anchura** del modelo. Cada una de estas unidades que actúa de forma paralela, representa una neurona que recibe información de otras unidades (neuronas) y calcula su propio valor de activación.

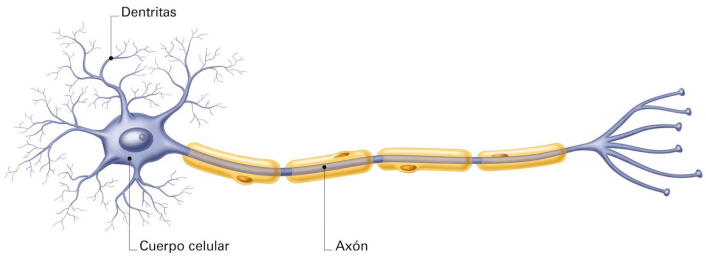


Figura 6.1: Representación de una neurona biológica.

El símil con las neuronas de los seres vivos es que estas se componen de dentritas que se encargan de recoger los estímulos de otras neuronas (entradas), un núcleo para su almacenamiento (activación) y un axón con terminales que le permiten transmitir los impulsos a otras neuronas (salidas).

La neurona artificial se define como en la Figura 6.2, donde identificamos una neurona biológica con una función $f : \mathbb{R}^n \to \mathbb{R}$ donde n es el número de unidades en la capa anterior (entradas).

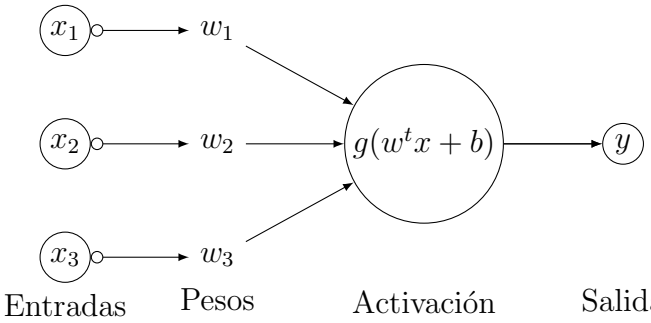


Figura 6.2: Representación de una neurona artificial de 3 entradas con función de activación g.

En conclusión, una red neuronal artifical estaría formada por un número arbitrario de capas y un número arbitrario de unidades (neuronas artificiales) en cada capa.

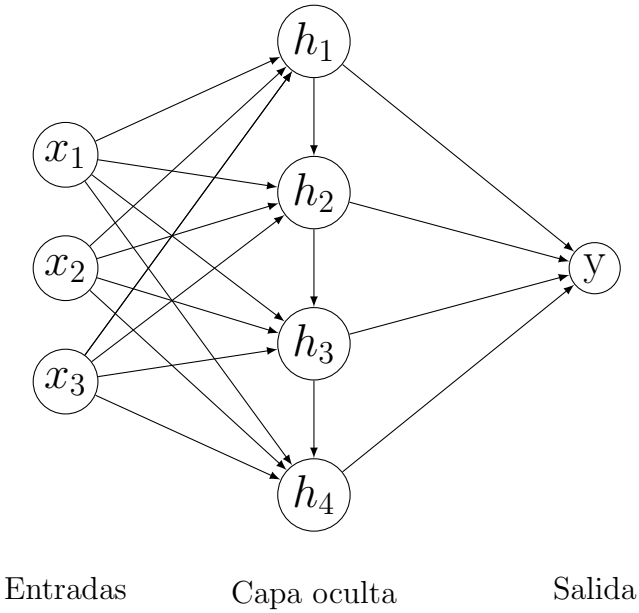


Figura 6.3: Representación de una neurona artificial de 3 entradas con función de activación g.

Observamos en la Figura 6.3 un ejemplo simplificado de una red neuronal con tres capas: capa de entrada, una capa oculta y capa de salida. La capa

de entrada tiene tres unidades, la oculta cuatro y la de salida una.

Una forma de entender las redes neuronales prealimentadas es conocer las limitaciones de los modelos lineales y considerar cómo superarlas. El éxito de los modelos lineales como la regresión logística o lineal radica en su eficiencia y fiabilidad. Sin embargo, no se puede ignorar el inconveniente de que están limitados a funciones lineales, por tanto no contemplan las posibles relaciones entre dos variables de entrada.

Para extender estos modelos de forma que representen funciones no lineales de x, podemos considerar un modelo lineal sobre una transformación de la entrada $\phi(x)$ donde ϕ es una transformación no lineal. El problema se traduce, entonces, en encontrar qué transformación no lineal ϕ aplicar a la entrada x:

- La primera opción sería estudiar manualmente qué función se ajusta más al problema concreto, lo que conllevaría un alto conocimiento del problema.
- Otra opción sería utilizar una función genérica ϕ de alta dimensionalidad para que siempre tenga la suficiente capacidad de adaptarse al conjunto de entrenamiento, lo cual no proporciona una buena generalización en el conjunto de test.
- El enfoque que aporta el *deep learning* es aprender ϕ en un conjunto de funciones parametrizadas. Así, tenemos el modelo

  $$y = f(x; \theta, w) = \phi(x; \theta)^T w$$

  donde θ es un conjunto de parámetros utilizado para aprender ϕ de entre la clase de funciones consideradas.

Este modelo nos aporta una gran flexibilidad pues la elección de la función de transformación depende del conjunto de funciones que estemos considerando. Además, al igual que en los modelos lineales deberemos predefinir ciertos aspectos de diseño: optimizador, función de coste y forma de las unidades de salida. Las redes neuronales prealimentadas introducen el concepto de capa oculta, por lo que tendremos que elegir también las funciones de activación que se utilizarán para calcular la salida de cada una de estas capas. También será necesario definir la arquitectura de la red, definiendo cuántas capas añadir y cuántas unidades tendrá cada una.

A continuación estudiaremos algunas de las funciones comentadas.

6.1.1 Función de coste

La principal diferencia entre los algoritmos lineales y las redes neuronales es la ya comentada no linealidad. Esta característica hace especialmente interesante considerar funciones de coste no convexas en este tipo de modelos.

Aunque en algunos casos podemos definir la función de coste como un estadístico de y condicionado a x. En la mayoría de los diseños, el modelo paramétrico definido define una distribución de probabilidad $p(y|x;\theta)$ y podemos aplicar el principio de máxima verosimilitud. Esto significa que usamos la entropía cruzada (Sección 4.8) entre los datos de entrenamiento y las predicciones del modelo como función de coste:

$$H(\theta) = -\mathbb{E}_{x,y\sim\hat{p}_{data}} \log p_{model}(y|x)$$

La ventaja de esta función de coste es que queda determinada en función de $p_{model}(y|x)$ ahorrándonos el trabajo de definir una función de coste propia para cada problema. Además, un tema recurrente en el diseño de redes neuronales es que el gradiente de la función de coste debe ser grande para evitar que el valor del gradiente vaya rápidamente a cero y nuestra función de coste cumple este requisito.

6.1.2 Unidades de salida

La elección de la función de coste está estrechamente relacinada con la elección de la unidad de salida. La mayoría de las veces simlpemente utilizamos la entropía cruzada de la distribución de los datos y de la distribución del modelo.

Aunque en esta sección nos centremos en unidades de salida, estas funciones pueden ser utilizadas en las capas ocultas también. Para unificar la notación supondremos que la red neuronal produce una salida de las capas ocultas definida por el vector de características $h = f(x;\theta)$.

Aunque existen muchas posibilidades, las principales unidades de salida son de los siguientes tipos:

Unidades lineales

Un tipo de unidad de salida está basada en una transformación lineal. Esto es, dado el vector de características h, la capa produce como salida el vector

$$\hat{y} = W^T h + b$$

El uso principal de este tipo de unidades de salida es producir la media de una distribución condicional normal:

$$p(y|x) = \mathcal{N}(y; \hat{y}, I)$$

Para esta probabilidad condicionada concreta, calcular la entropía cruzada (log-verosimilitud negativa) es equivalente a minimizar usando el error cuadrático medio.

Unidades sigmoidales

Este tipo de unidades se utilizan para tareas que requieren la predicción de una variable binaria y. La probabilidad condicionada aplicada en la técnica de máxima verosimilitud en este caso se correspondería con la definición de una distribución de Bernouilli sobre y condicionada a x. Esta distribución se define con un número en el intervalo $[0, 1]$ determinado por la probabilidad $P(y = 1|x)$.

Para cumplir la restricción de permanecer en el intervalo $[0, 1]$ podríamos considerar la función de probabilidad

$$P(y = 1|x) = \text{máx}\{0, \text{mín}\{1, w^T h + b\}\}$$

Sin embargo, esto tiene un problema pues cada vez que el valor de la expresión $w^T h + b$ quedara fuera del intervalo, el gradiente valdría 0, lo cual supondría un mal funcionamiento del método.

Para asegurar un buen funcionamiento del método, asegurándonos de que el gradiente no es muy cercano a cero se utiliza un enfoque diferente. Con-

siste en utilizar una salida sigmoidal combinada con el criterio de máxima verosimilitud.

Así, a salida de una unidad de salida sigmoidal estaría definida por

$$\hat{y} = \sigma(w^T h + b) = \frac{1}{1 + e^{-(w^T h + b)}}$$

Podemos ver esta salida como una unidad que tiene dos componentes:

1. En primer lugar se aplica una capa lineal para calcular $z = w^T h + b$
2. En segundo lugar se aplica una **función de activación sigmoidal** para convertir z en una probabilidad.

Para construir la distribución de probabilidad sobre y podemos partir de una distribución de probabilidad $\tilde{P}(y)$ no normalizada (no suma 1). A partir de aquí, podemos obtener una distribución de probabilidad dividiendo entre la constante adecuada y que $y \in \{0, 1\}$, obteniendo:

$$log\tilde{P}(y) = yz,$$

$$\tilde{P}(y) = e^{yz},$$

$$P(y) = \frac{e^{yz}}{e^{yz} + e^{(1-y)z}} = \frac{1}{1 + \frac{e^z}{e^{2yz}}} = \frac{1}{1 + e^{(1-2y)z}},$$

$$P(y) = \sigma((2y - 1)z)$$

Hemos obtenido que la función de distribución sobre y es una Bernouilli determinada por la transformación logística de z.

Susituyendo esta función de probabilidad en la fórmula de la entropía cruzada obtenemos:

$$J(\theta) = -\log P(y|x) = -\log \sigma((2y - 1)z)$$

que está bien definida pues σ está definida en $(0, 1)$ abierto por lo que su logaritmo es finito y bien definido.

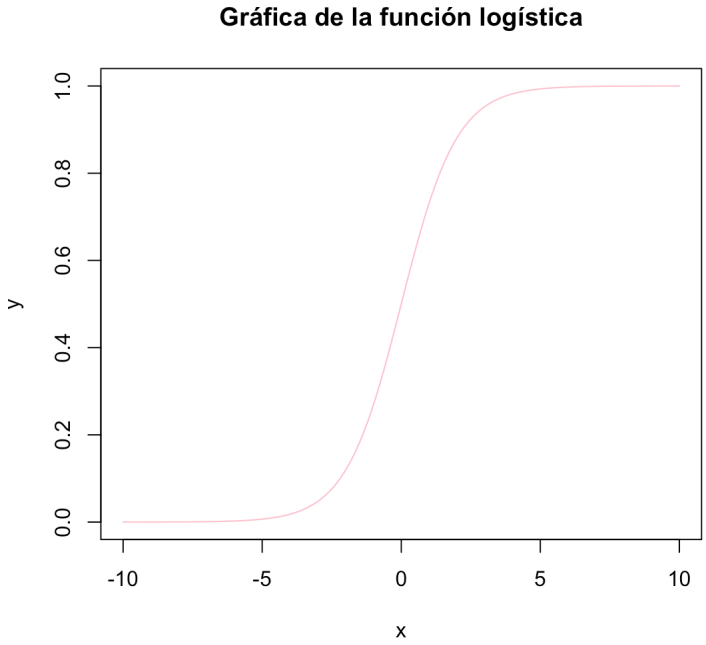


Figura 6.4: Gráfica de la función logística. Toma valores en (0,1)

Unidades con activación softmax

Cada vez que queramos representar una distribución de probabilidad sobre una variable discreta que toma n valores diferentes, usaremos la **función de activación *softmax***. Podemos ver esta función como una generalización de una función sigmoidal.

En la práctica, este tipo de unidades de salida son muy utilizadas en los algoritmos de clasificación, para representar la distribución de probabilidad de pertenecer a las diferentes n clases. De forma extraordinaria también se pueden utilizar en las capas intermedias del modelo si queremos que el modelo elija entre una de las n diferentes opciones en alguna variable interna.

Para generalizar el caso anterior a una variable discretade n valores, necesitamos producir un vector $\hat{y}$ con $\hat{y}_i = P(y = i|x)$. En este caso se añade a la condición de que cada $\hat{y}_i$ esté entre 0 y 1 que la suma del vector sea 1 para que represente una distribución de probabilidad.

Utilizando la misma filosofía que en el caso anterior definimos una probabilidad logarítmica no normalizada

$$z = W^T h + b$$

donde $z_i = \log \tilde{P}(y = i|x)$.

Pasando a exponencial y normalizando obtenemos la función *softmax*

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

De nuevo, la función de coste se obtendría sustituyendo el valor en la función de entropía cruzada.

6.1.3 Unidades ocultas

Aunque cualquiera de las funciones de activación comentadas en la sección anterior se pueden aplicar en las unidades de una capa oculta, existen otras más adecuadas para ello. De hecho, el diseño de las unidades ocultas es un campo de investigación muy activo. A lo largo de esta sección desarrollaremos algunas de las más comunes.

Unidades lineales rectificadas (ReLu) y generalizaciones

Las unidades lineales rectificadas [9] usan como función de activación

$$g(z) = \text{máx}\{0, z\}$$

Estas unidades son fácil de optimizar dado que son muy similares a las unidades lineales. La única diferencia radica en que la unidad lineal rectificada toma el valor 0 en la mitad del su dominio. Observamos la gráfica en la Figura 6.5.

Existen muchas modificaciones de las unidades lineales rectificadas. Una desventaja de este tipo de unidades es que no pueden aprender usando

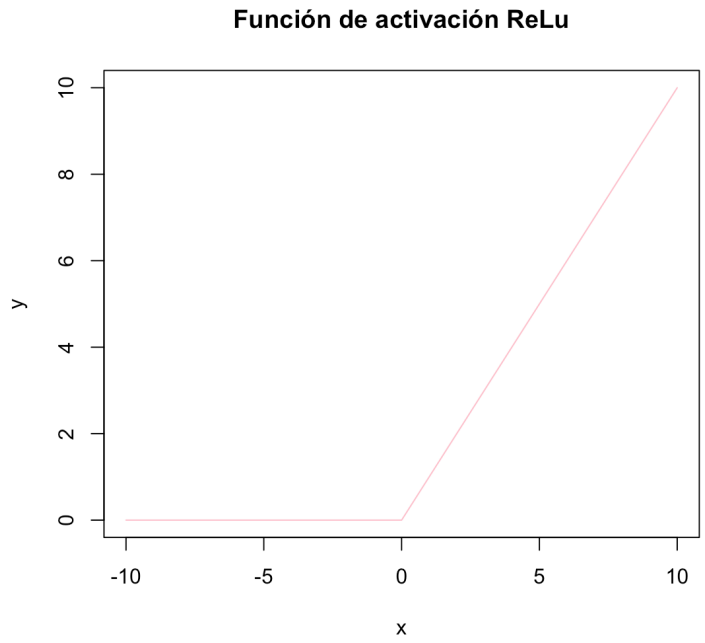


Figura 6.5: Gráfico de la función de activación de las unidades lineales rectificadas.

métodos basados en gradiente en los ejemplos cuya activación sea 0. Muchas de las 8 generalizaciones intentan solventar este problema. Definen $h_i = g(z, \alpha)_i = \text{máx}(0, z_i) + \alpha_i \,\text{mín}(0, z_i)$ y varían en el valor de las α_i:

1. **Rectificación valor absoluto**: fija $\alpha_i = -1$ para obtener $g(z) = |z|$.
2. ***Leaky* ReLu:** fijan α_i a un valor pequeño por ejemplo 0,01.
3. **ReLu paramétrica:** utiliza valores de α_i no fijos, paramétricos.

Sigmoidal y tangente hiperbólica

Antes de la introducción de las unidades lineales rectificadas, la mayoría de las redes neuronales usaban la función sigmoidal de activación

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

cuya gráfica podemos observar en la Figura 6.4 o la tangente hiperbólica como función de activación

$$g(z) = \tanh(z)$$

cuya gráfica es

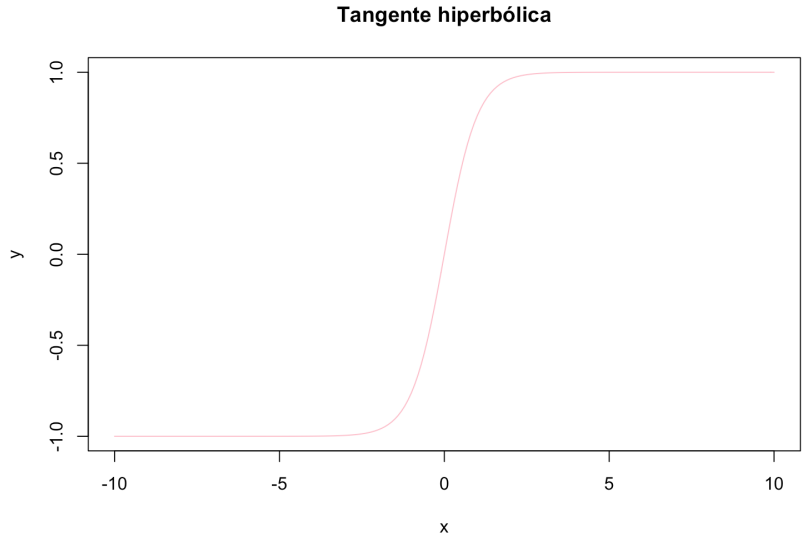


Figura 6.6: Gráfica de la tangente hiperbólica.

Ambas funciones de activación están relacionadas mediante la fórmula $\tanh(z) = 2\sigma(2z) - 1$.

Capítulo 7

Redes Neuronales Convolucionales

[4] Las Redes Neuronales Convolucionales o *CNN* son un tipo especializado de red neuronal para el procesamiento de datos con estructura de cuadrícula específica.

Durante el capítulo describiremos la operación matemática en la que se basan este tipo de redes, conocida como convolución. Posteriormente describiremos otra operación que suelen implementar este tipo de redes, el *pooling*.

7.1 La operación de Convolución

Una convolución es una operacion en dos funciones de argumentos en $\mathbb{R}$ generando una tercera función combinando las dos anteriores.

Dadas dos funciones que toman valores reales $f, g : \mathbb{R} \to \mathbb{R}$ con la regularidad necesaria, se denota como $(f * g)$ y se define como el producto de ambas funciones habiendo previamente invertido y desplazado una de ellas. Esto es:

$$s(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(s)g(t-s)ds \tag{7.1}$$

En la terminología de las redes neuronales convolucionales, el primer argumento (f) es conocido como **entrada** y el segundo (g) como **núcleo**. La salida se denomina **mapa de características**.

Para la definición de convolución hemos supuesto f y g funciones continuas definidas en $\mathbb{R}$. Sin embargo, esta condición de proporcionar valores en cada instante de tiempo (de forma continua) no es realista. Cuando trabajamos con datos en un ordenador, el tiempo se discretiza y nuestras funciones pasarían a tomar valores de forma discreta. Para ello, definimos la **función de convolución discreta,** que será la utilizada por las redes neuronales convolucionales de la siguiente manera:

$$s(t) = (f * g)(t) = \sum_{s=-\infty}^{\infty} f(s)g(t-s) \tag{7.2}$$

En los modelos de aprendizaje, la entrada suele ser un vector multidimensional de datos y el núcleo un vector multidimensional de parámetros que se adaptan mediante el algoritmo de aprendizaje. Supondremos que las funciones toman valor 0 en todo su dominio menos en el conjunto finito de puntos que se almacenan como valores. Esto significa que, en la práctica, podemos implementar la suma infinita de la definición como una suma finita de elementos ignorando aquellos elementos que sumen una cantidad nula.

Normalmente se utilizan más de una convolución al mismo tiempo en diferentes ejes.

Por ejemplo, si consideramos una imagen bidimensional I como entrada, podríamos aplicar un núcleo bidimensional K de la siguiente manera:

$$\begin{aligned} S(i,j) = (I * K)(i,j) &= \sum_m \sum_n I(m,n)K(i-m, j-n) \\ &= \sum_m \sum_n K(m,n)I(i-m, j-n) \end{aligned} \tag{7.3}$$

7.1.1 Ejemplo de convolución

En esta subsección veremos un ejemplo de convolución sencilla para entender mejor su funcionamiento.

Por ejemplo, consideremos el último caso estudiado en la sección anterior. Tenemos una imagen bidimensional $I \in \mathcal{M}_{nxm}(\mathbb{R}) = \mathcal{M}_{5x5}(\mathbb{R})$ y aplicamos un filtro K de dimensiones $(i, j) = (2, 3)$ en los respectivos ejes. El resultado sería entonces un mapa de características de dimensiones $(n-i+1, m-j+1) = (4, 3)$.

Veamos el funcionamiento de la convolución gráficamente. Tenemos la siguiente entrada:

$I_{(1,1)}$	$I_{(1,2)}$	$I_{(1,3)}$	$I_{(1,4)}$	$I_{(1,5)}$
$I_{(2,1)}$	$I_{(2,2)}$	$I_{(2,3)}$	$I_{(2,4)}$	$I_{(2,5)}$
$I_{(3,1)}$	$I_{(3,2)}$	$I_{(3,3)}$	$I_{(3,4)}$	$I_{(3,5)}$
$I_{(4,1)}$	$I_{(4,2)}$	$I_{(4,3)}$	$I_{(4,4)}$	$I_{(4,5)}$
$I_{(5,1)}$	$I_{(5,2)}$	$I_{(5,3)}$	$I_{(5,4)}$	$I_{(5,5)}$

Tras aplicar la convolución con las dimensiones del núcleo indicadas, denotando a S como la función convolución tenemos:

S(1,1)	S(1,2)	S(1,3)	S(1,4)
S(2,1)	S(2,2)	S(2,3)	S(2,4)
S(3,1)	S(3,2)	S(3,3)	S(3,4)

donde $S(1,1) = f(I_{(1,1)}, I_{(1,2)}, I_{(2,1)}, I_{(2,2)}, I_{(3,1)}, I_{(3,2)})$ y así sucesivamente.

7.2 *Pooling*

Una típica capa de una red neuronal consiste en tres pasos:

1. La capa realiza varias convoluciones en paralelo para producir un conjunto de activaciones lineales.
2. Al resultado de cada activación lineal se le aplica una función de activación no lineal.
3. Se usa una función de *pooling* para modificar aún más la salida de la capa.

Una función de *pooling* reemplaza la salida de la red en una determinada localización con valores estadísticos de las salidas cercanas. Como ejemplos de funciones de *pooling* podemos destacar el ***max pooling*** que devuelve el máximo de un vecindario rectangular, o aquella que devuelve la media o la norma L^2 de vecindarios rectangulares.

En todos los casos, el aplicar *pooling* hace la representación prácticamente invariante frente a pequeñas variaciones en la entrada, consiguiendo más capacidad de generalización.

Capítulo 8

Modelado de secuencias: Redes Neuronales Recurrentes

Al igual que en el capítulo anterior, en este vamos a desarrollar una clase concreta de redes neuronales, las redes neuronales recursivas y recurrentes. Esta familia de redes neuronales se utilizan para el procesamiento de datos secuencial, por lo que será natural su uso en el problema que se trata en este proyecto.

8.1 Motivación

La innovación con respecto a las redes neuronales convencionales será el aplicar a estas nuevas técnicas de *deep learning* una de las primeras ideas que surgieron en el aprendizaje automático y los modelos estadísticos: compartir los parámetros a lo largo de todo el aprendizaje. El hecho de compartir parámetros durante el aprendizaje del modelo hará posible la extensión y aplicación del modelo a ejemplos de diferentes longitudes y generalizar a través de ellos.

Podemos entender la importancia del hecho de compartir parámetros durante el modelo en un ejemplo sencillo de procesamiento del lenguaje natural. Supongamos las frases [1] y [2]:

[1] Estuve en Nepal en 2017.
[2] En 2017 estuve en Nepal.

Si quisiéramos diseñar un algoritmo de aprendizaje automático que reconociera en qué año estuvo el narrador en Nepal, debería reconocerse el año 2017 como la parte importante de información, independientemente de que aparezca en la quinta o en la segunda palabra de la oración. Una red neuronal tradicional usaría parámetros independientes para cada entrada, por lo que necesitaría aprender todas las reglas del lenguaje de forma independiente para cada posición en la frase. Este problema se solventaría considerando una red neuronal recurrente que comparta los mismos pesos a lo largo de todos los pasos necesarios durante el aprendizaje.

Para simplificar la notación durante el capítulo consideraremos que la red actúa sobre una secuencia que contiene vectores $x^{(t)}$ aunque en la práctica las redes suelen operar sobre conjuntos de ellas (*minibatches*).

Al igual que en Capítulo 7 se definía la estructura del modelo de las redes neuronales convolucionales como un grafo acíclico, la recurrencia de este tipo de redes ampliará el concepto a grafos cíclicos. Estos ciclos representarán la influencia del valor actual de la variable en su propio valor en el siguiente paso del aprendizaje.

8.2 Despliegue de Grafos Computacionales

Un **grafo computacional** [4] es una forma de formalizar la estructura de un conjunto de cálculos, como aquellos que involucran un conjunto de entradas y parámetros para producir una salida y pérdida.

En esta sección desarrollaremos la idea de **desplegar** un cálculo recurrente en un grafo computacional con su correspondiente estructura de cadena de eventos.

Por ejemplo, consideramos la fórmula clásica de un sistema dinámico:

$$s^{(t)} = f(s^{(t-1)}; \theta) \tag{8.1}$$

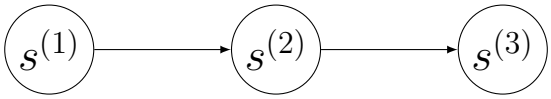
donde $s^{(t)}$ es el estado del sistema.

Es claro que esta fórmula es recurrente porque en la definición de un estado concreto de s en el tiempo t se utiliza el valor del mismo estado en el momento de tiempo $t-1$.

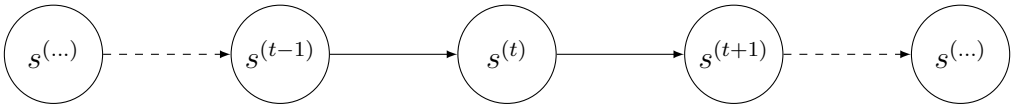
Si consideramos un número de pasos finito τ, el grafo puede ser desplegado aplicando la definición $\tau - 1$ veces. Por ejemplo, para $\tau = 3$ tenemos:

$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta) \tag{8.2}$$

Desplegando sucesivamente la fórmula con la definición de s hemos obtenido una expresión que no involucra recurrencia y que puede ser expresada con un gráfico acíclico. De esta forma el grafo para la Ecuación 8.2 sería:



Y de forma genérica, para la Ecuación 8.1 tendríamos el siguiente grafo



La mayoría de las redes neuronales recurrentes utilizan la Ecuación 8.3 para definir el valor de salida de las capas ocultas.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{8.3}$$

Cuando una red neuronal recurrente se entrena para una tarea que requiere predecir un resultado a partir de unos datos, aprende usando $h^{(t)}$ como un tipo de resumen de pérdida de los aspectos relevantes para la tarea de la secuencia de entradas hasta el momento t.

La Ecuación 8.3 se puede representar de dos formas diferentes:

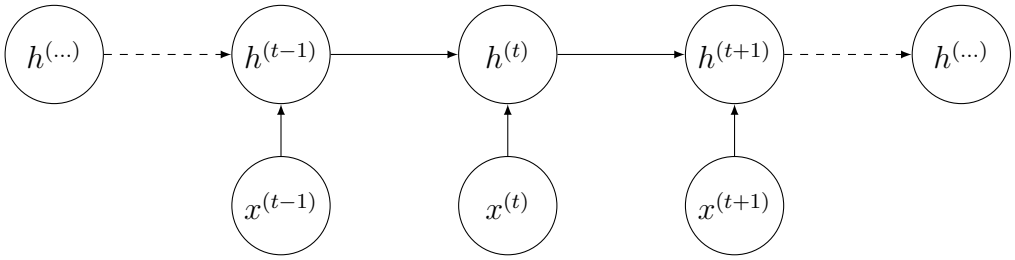


Figura 8.1: Representación extendida de las operaciones de las capas ocultas en una red neuronal recursiva.

1. Considerando un nodo por cada uno de los estados por los que pasa cada una de las variables del modelo como en la Figura 8.1.

2. Considerando una forma resumida, donde cada nodo representa las componentes que existirían en una implementación física del modelo como en la Figura 8.2.

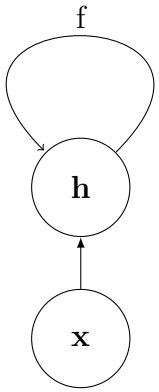


Figura 8.2: Representación esquematizada de las operaciones de las capas ocultas en una red neuronal recursiva.

Ambas formas de representación tienen sus ventajas, mientras que la segunda es compacta, la primera nos ofrece una descripción explícita de los cálculos que se llevarán a cabo. A lo largo del capítulo utilizaremos indiferentemente ambas notaciones.

8.3 Redes Neuronales Recurrentes

Una vez hemos introducido la idea de compartir parámetros durante todo el aprendizaje y el despliegue de grafos computacionales podemos diseñar una gran variedad de redes neuronales recurrentes.

Podemos observar un esquema de red recurrente en la Figura 8.3.

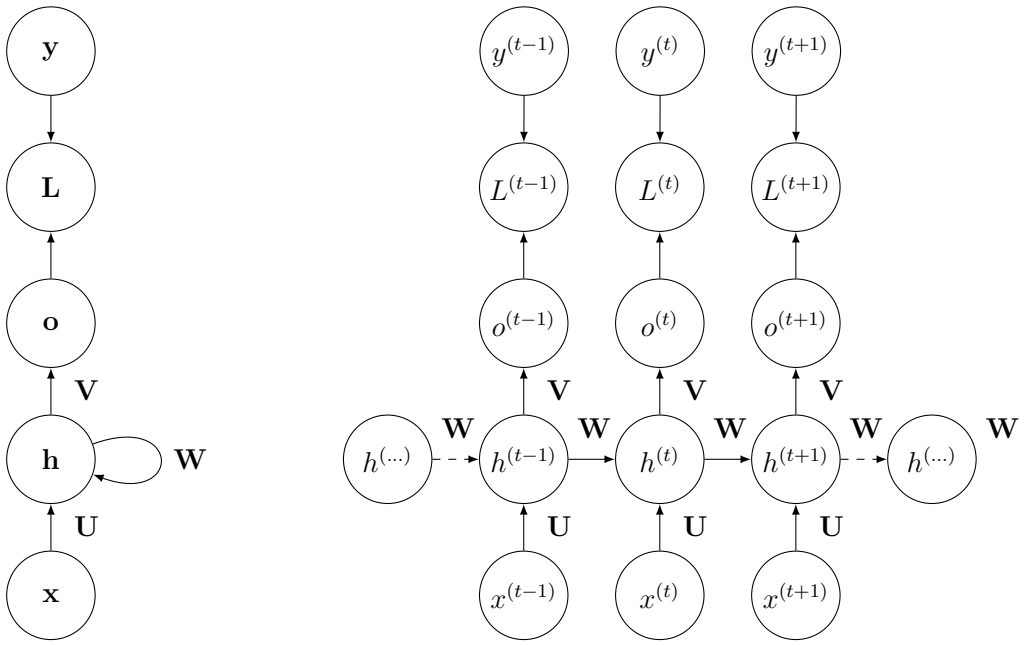


Figura 8.3: Representaciones de una red neuronal recurrente genérica.

En cuanto a la notación, establecemos como **x** los valores de entrada y **o** con la secuencia de valores de salida. Por su parte, una medida **L** determina cómo de acertado es cada valor de **o** para la correspondiente etiqueta de entrenamiento **y**. La entrada de las capas ocultas están parametrizadas por una matriz de pesos **U**, mientras que las conexiones entre capas ocultas recurrentes se parametrizan con la matriz de pesos **W** compartida, y la conexión de las capas ocultas con la salida está parametrizada por la matriz de pesos **V**.

Una vez establecida la arquitectura de la red, nos centramos en desarrollar las ecuaciones propagación hacia delante (*forward propagation*). En este caso supondremos que la función de activación de las capas ocultas será una

tangente hiperbólica (*tanh*) y que la salida es discreta. Una forma común de representar variables discretas es considerar la salida **o** como una probabilidad logarítmica no normalizada para cada posible valor de la variable discreta. Entonces, podemos aplicar una operación *softmax* a dichas probabilidades como se explica en la Sección 6.1.2.

Con todas estas consideraciones, tomando $t \in [0, \tau]$ podemos definir las siguientes ecuaciones:

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\ \mathbf{h}^{(t-1)} +\ \mathbf{U}\mathbf{x}^{(t)} \tag{8.4}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}) \tag{8.5}$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \tag{8.6}$$

$$\hat{\mathbf{y}}^{(t)} = softmax(\mathbf{o}^{(t)}) \tag{8.7}$$

donde los parámetros son los vectores de sesgo **b** y **c** junto con las matrices de pesos anteriormente comentadas **U**, **V** y **W**.

En cuanto a la función de pérdida para una secuencia de entrada **x** emparejada con una secuencia de valores **y** sería simplemente la suma de las pérdidas. Por ejemplo, considerando la entropía cruzada $\mathcal{L}^{(t)}$ de $y^{(t)}$ dados $\mathbf{x}^{(t)}, ..., \mathbf{x}^{(t)}$ entonces tenemos:

$$\mathcal{L}(\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, ..., \mathbf{y}^{(\tau)}\}) = \sum_t \mathcal{L}^{(t)}) = -\sum_t \log p(y^{(t)} | \mathbf{x}^{(1)}, ..., \mathbf{x}^{(t)}\}) \tag{8.8}$$

Calcular el gradiente de esta función de pérdida con respecto a los parámetros es una operación costosa. Su cálculo require de una propagación hacia delante de izquierda a derecha seguida de una propagación hacia atrás de derecha a izquierda en nuestro grafo. El tiempo de ejecución sería $\mathcal{O}(\tau)$ y no se puede reducir con paralización porque el grafo es inherentemente secuencial, cada paso de tiempo tiene que ser ejecutado después de haber ejecutado el

anterior. Los cálculos realizados en la propagación hacia delante se almacenan utilizándose en la propagación hacia atrás, así conseguimos una coste de memoria de $\mathcal{O}(\tau)$ nuevamente.

El algoritmo de propagación hacia atrás (*backpropagation*) utilizado recorrer el grafo en $\mathcal{O}(\tau)$ se llama ***back-propagation through time*** **(BPTT)** o retro-propagación a través del tiempo y lo estudiamos en la siguiente sección.

8.3.1 Cálculo del Gradiente

El cálculo del gradiente en una red neuronal recurrente es sencillo, basta con aplicar el algoritmo *back-propagation* clásico al gráfico extendido.

Para entender mejor el comportamiento del método $BPTT$, lo aplicaremos sobre las Ecuaciones 8.4 a la 8.7:

Para cada nodo $\mathbf{N}$ necesitamos calcular el gradiente $\nabla_{\mathbf{N}} L$ recursivamente basándonos en el gradiente calculado en los nodos que lo siguen en el grafo.

Empezamos la recurrencia con los nodos que siguen inmediantamente a la pérdida final:

$$\frac{\partial L}{\partial L^{(t)}} = 1 \tag{8.9}$$

Asumiendo que las salidas $\mathbf{o}^{(t)}$ se usan como argumento para una función *softmax* que nos proporciona el vector de probabilidades $\hat{\mathbf{y}}$ sobre la salida, el gradiente $\nabla_{\mathbf{o}^{(t)}} L$ de las salidas en el paso de tiempo t, para todo i, t es:

$$(\nabla_{\mathbf{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i(t)} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i(t)} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}} \tag{8.10}$$

Como estamos cálculando hacia atrás, empezando por el final de la secuencia. En el paso final τ, $\mathbf{h}^{\tau}$ solo tendría $\mathbf{o}^{\tau}$ como descendiente, por lo que su gradiente es sencillo:

$$\nabla_{\mathbf{h}^{(\tau)}} L = \mathbf{V}^T \nabla_{\mathbf{o}^{(\tau)}} L \tag{8.11}$$

Iterando ahora de forma descendente desde $t = \tau - 1$ hasta $t = 1$, observamos que $\mathbf{h}^{(t)}$ tendría como descendientes a $\mathbf{o}^{(t)}$ y $\mathbf{h}^{(t-1)}$. Entonces, el gradiente se calcula de la siguiente manera:

$$\nabla_{\mathbf{h}^{(t)}} L = \left(\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}}\right)^T (\nabla_{\mathbf{h}^{(t+1)}} L) + \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}}\right)^T (\nabla_{\mathbf{o}^{(t)}} L) = \tag{8.12}$$

$$= \mathbf{W}^T (\nabla_{\mathbf{h}^{(t+1)}} L) diag\left(1 - (\mathbf{h}^{(t+1)})^2\right) + \mathbf{V}^T (\nabla_{\mathbf{o}^{(t)}} L)$$

Donde $\text{diag}\left(1 - (\mathbf{h}^{(t+1)})^2\right)$ se corresponde con la matriz diagonal cuyos elementos son $1 - (\mathbf{h}_i^{(t+1)})^2$ que es la matriz Jacobiana de la tangente hiperbólica asociada a la unidad oculta i en el momento de tiempo $t + 1$.

Una vez que hemos obtenido el gradiente de los nodos internos del grafo de cómputo, podemos obtener los gradientes en los nodos de parámetros. A la hora de calcular el gradiente de esas variables tenemos que tener en cuenta que los parámetros se comparten a lo largo de los pasos. Usando $\nabla_{\mathbf{W}^{(t)}}$ para notar la contribución de los pesos en el instante de tiempo t al gradiente, las expresiones de los gradientes del resto de parámetros son:

$$\nabla_{\mathbf{c}} L = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}} L = \sum_t \nabla_{\mathbf{o}^{(t)}} L \tag{8.13}$$

$$\nabla_{\mathbf{b}} L = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}} L = \sum_t diag\left(1 - \left(\mathbf{h}^{(t)}\right)^2\right) \nabla_{\mathbf{h}^{(t)}} L \tag{8.14}$$

$$\nabla_{\mathbf{v}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_o^{(t)}}\right) \nabla_{\mathbf{v}} o_i^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}} L) \mathbf{h}^{(t)^T} \tag{8.15}$$

$$\nabla_{\mathbf{W}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\mathbf{W}^{(t)}} h_i^{(t)} = \sum_t diag\left(1 - \left(\mathbf{h}^{(t)}\right)^2\right) (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{h}^{(t-1)^T} \tag{8.16}$$

$$\nabla_{\mathbf{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\mathbf{U}^{(t)}} h_i^{(t)} = \sum_t diag\left(1 - \left(h^(t)\right)^2\right) (\nabla_{\mathbf{h}^t} L) \mathbf{x}^{(t)^T} \tag{8.17}$$

8.3.2 El problema de las dependencias a largo plazo

El problema de aprender dependencias a largo plazo en las redes recurrentes radica en que el gradiente propagado durante muchas etapas tiende a desaparecer (la mayoría de las veces) o a explotar. Incluso si asumimos que los parámetros del modelo son aquellos que hacen a la red estable, la dificultad de las dependencias a largo plazo surgen de los pesos exponencialmente más pequeños dados en las iteracciones a largo plazo (involucrando multiplizaciones de varias matrices Jacobianas) en comparación con las de corto plazo. Durante esta sección expondremos el problema con más precisión.

Las redes recurrentes involucran la composición de la misma función múltiples veces, una por paso. Estas composiciones pueden provocar un comportamiento extremadamente alejado de lo lineal. En particular, la composición de funciones empleada en las redes neuronales recurrentes son similares a la multiplicación de matrices. Podemos pensar en la relación recurrente

$$\mathbf{h}^{(t)} = \mathbf{W}^T \mathbf{h}^{(t-1)} \tag{8.18}$$

como una red neuronal recurrente muy simple sin función de activación y sin entradas. Podemos simplificar esta expresión de la forma

$$\mathbf{h}^{(t)} = (\mathbf{W}^t)^T \mathbf{h}^{(0)}, \tag{8.19}$$

y si $\mathbf{W}$ admite descomposición en valores propios de la forma

$$\mathbf{W} = \mathbf{Q} \Lambda \mathbf{Q}^T \tag{8.20}$$

con $\mathbf{Q}$ matriz ortogonal, la recurrencia podría ser simplificada aún más

$$\mathbf{h}^{(t)} = \mathbf{Q}^T \Lambda^t \mathbf{Q} \mathbf{h}^{(0)} \tag{8.21}$$

Los valores propios se elevan a la potencia t, produciendo valores propios con magnitud menos de uno para tender desvanecerse y valores propios con magnitud mayor que uno para explotar. Así, toda componente de $\mathbf{h}^{(0)}$ que no esté alineada con el mayor valor propio tenderá a ser descartada.

8.3.3 Redes neuronales recurrentes con memoria a corto-largo plazo

Las redes neuronales recurrentes con memoria a corto-largo plazo o *Long Short-Term Memory (LSTM)* tratan de solucionar el problema de las dependencias a largo plazo 8.3.2.

Estas redes son el ejemplo canónico de un subgrupo de redes neuronales, las **redes neuronales recurrentes con puertas** o ***gated RNNs***. Se basan en la idea de crear caminos a través del tiempo que tengan derivadas que no se desvanezcan ni exploten.

Las *LSTM* surgen de la idea de introducir auto-bucles para producir caminos donde el gradiente puede fluir durante largos períodos de tiempo. Además, introduen la innovación de que los pesos del auto-bucle estén condicionados al contexto. Al controlar los pesos del auto-bucle a través otra unidad oculta, la escala de tiempo de integración puede cambiar dinámicamente. Es decir, incluso para una *LSTM* de parámetros fijos, la escala de tiempo de integración podría cambiar en función de la secuencia de entrada. Las *LSTM* han producido resultados exitosos en muchas aplicaciones, entre ellas el Procesamiento del Lenguaje Natural (PLN), motivo por el que las desarrollamos durante esta memoria.

En la Figura 8.4 se esquematiza el diagrama de bloque que representa una celda de una *LSTM*. Las celdas se conectan entre ellas recurrentemente, reemplazando las unidades ocultas de las redes neuronales comunes. Estas celdas tienen una recurrencia interna (*self-loop*) además de la recurrencia interna entre celdas. Cada celda tiene las mismas entradas y salidas que una red recurrente normal, pero tiene más parámetros y sistemas de puertas que controlan el flujo de información.

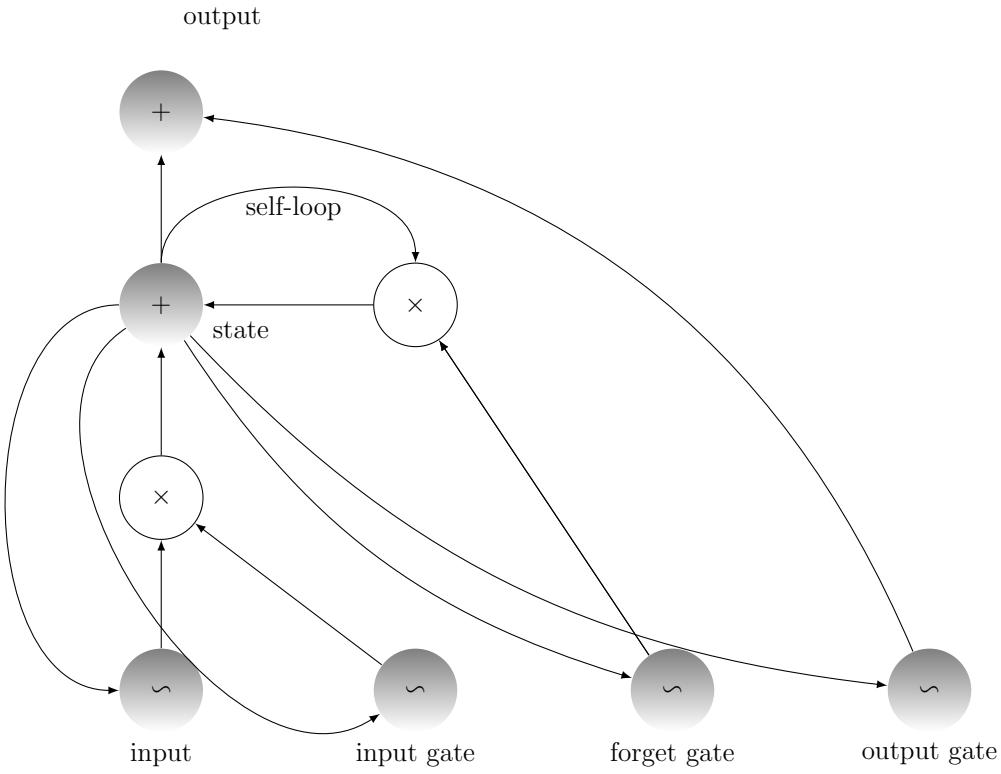


Figura 8.4: Representaciones de una red neuronal recurrente genérica.

La componente más importante de esta estructura es el estado $s_i^{(t)}$, el cual tiene una auto-bucle lineal. Los pesos de este auto-bucle están determinados por una unidad de olvido o *forget gate unit* $f_i^{(t)}$, la cual fija los pesos a un valor en $[0, 1]$ mediante una unidad sigmoidal:

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right), \tag{8.22}$$

donde $\mathbf{x}^{(t)}$ es el actual vector de entrada y $\mathbf{h}^{(t)}$ es el vector de la actual capa oculta, que contiene las salidas de todas las celdas previas y $\mathbf{b}^{(t)}$, $\mathbf{U}^{(t)}$ y $\mathbf{W}^{(t)}$ son los sesgos, pesos de entrada y pesos recurrentes de las unidades de olvido respectivamente.

El valor del estado interno de la celda se actualiza mediante la siguiente fórmula:

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right) \tag{8.23}$$

La unidad de entrada externa o *external input gate unit* $g_i^{(t)}$ se calcula de forma similar a la unidad de olvido (mediante una unidad sigmoidal) pero con sus propios parámetros:

$$g_i^{(t)} = \sigma \left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)} \right) \tag{8.24}$$

La salida $h_i^{(t)}$ de la celda se puede expresar en función de la unidad de salida $q_i^{(t)}$ que también utiliza una unidad sigmoidal:

$$h_i^{(t)} = \tanh \left(s_i^{(t)} \right) q_i^{(t)}, \tag{8.25}$$

$$q_i^{(t)} = \sigma \left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)} \right) \tag{8.26}$$

cuyos parámetros $\mathbf{b}^o$, $\mathbf{U}^o$ y $\mathbf{W}^o$ son los sesgos, pesos de entrada y pesos recurrentes respectivamente.

Las redes neuronales a corto-largo plazo han mostrado aprender las dependencias a largo plazo más fácilmente que las arquitecturas recurrentes dado que solucionan su principal inconveniente, por lo que serán las que se utilizarán en la práctica.

Sin embargo, por la definición de la recurrencia solo se comparten parámetros en un sentido (desde el comienzo de la secuencia hasta el final). Puede ser de interés para algunos problemas concretos que se compartan parámetros en ambos sentidos.

Por ejemplo, en el caso de la extracción de entidades o de cualquier problema relacionado con el PLN es esencial el contexto para cada palabra, por lo que cobraría sentido modelar este tipo de secuencias con parámetros compartidos en ambos sentidos (dos recurrencias). De esta idea surgen las **biLSTM** del inglés *bidirectional LSTM*. Estas redes representan una nueva arquitectura en las que cada capa estará formada por dos capas recurrentes en sentidos opuestos. No es más que una extensión de la *LSTM* por lo que su funcionamiento se puede deducir del resto de la Sección.

Capítulo 9

Etiquetado de secuencias

En este capítulo haremos una breve introducción al etiquetado de secuencias y el por qué de su importancia en este proyecto. Para ello, introduciremos conceptos que hemos utilizado para su implementación como los *Conditional Random Fields* o campos condicionales aleatorios (CRF) y el *algoritmo de Viterbi* [15] .

9.1 Motivación

Es fundamental para muchas aplicaciones tener la habilidad de predecir múltiples variables que dependan unas de otras. Por ejemplo la clasificación regiones en una imagen y el procesamiento de lenguaje natural. En estas aplicaciones, necesitamos predecir un vector salida $y = \{y_0, y_1, ..., y_T\}$ de variables aleatorias dado un vector de características x.

En el enfoque de este problema de predicción multivariante, especialmente cuando es nuestro objetivo maximizar el número de etiquetas y_s que están correctamente etiquetadas, es aprender un clasificador por posición que asigne $x \rightarrowtail y_s$ para cada s. La dificultad, sin embargo, radica en que las variables tengan dependencias complejas. Por ejemplo, en la identificación del tipo de palabra en español sabemos que los adjetivos suelen ir tras un sustantivo. Otra dificultad es que las variables de salida representen una estructura más compleja.

Es natural representar la manera en la cual las variables de salida depende unas de otras con modelos basados en grafos, como por ejemplo las redes neuronales. Estos modelos de enfoque *generativo* intentan modelar una distribución de probabilidad conjunta $p(x, y)$ sobre las entradas y salidas. Aunque este enfoque tenga obvias ventajas, puede presentar la dificultad de que la dimensionalidad de x sea inabarcable.

Una posible solución es el enfoque *discriminativo*. de esta forma, se modela la distribución condicional $p(y \mid x)$ directamente que es lo que necesitamos para el problema de clasificación. Este es el enfoque que toman los CRFs.

Los CRFs son, esencialmente, una forma de combinar las ventajas de la clasificación discriminativa con los modelos basados en grafos.

9.2 Modelado de secuencias

Mientras que un clasificador puede predecir una sola variable de clase, el poder de los modelos basados en grafos radica en la habilidad de modelar varias variables que son independientes. En esta sección discutimos la forma más simple de dependencia, aquella en la que las variables de salida del modelo basado en gráficos están organizadas en una secuencia. Motivamos esta introducción con nuestro problema concreto de reconocimiento de entidades, donde podemos encontrarnos entidades formadas por una sola palabra como *casa* o por varias de ellas como *Banco de España*.

Una primera aproximación al identificador de entidades sería clasificar cada palabra de forma independiente. El problema de este enfoque es que asume que dada una entrada, todo el etiquetado de salida es independiente. En la realidad esto no se verifica.

Una forma de disminuir la suposición de independencia es organizar las variables de salida en una cadena lineal. Este enfoque es el tomado por los *HMM* (ver Sección 4.7).

Un *HMM* modela una secuencia de observaciones $X = x_{t\,t=1}^{T}$ asumiendo que hay una secuencia subyacente de estados $Y = y_{t\,t=1}^{T}$. Notamos S el conjunto finito de posibles estados y O el conjunto finito de posibles observaciones. En nuestro problema concreto, cada observación x_t se identifica con una palabra

en la posición t, y cada estado y_t con una etiqueta IOB.

Para modelar la distribución conjunta $p(Y, X)$ de forma tratable, un *HMM* hace dos suposiciones de independencia:

1. Cada estado depende solamente de su predecesor inmediato.
2. Cada observación de la variable x_t depende sólo del estado actual de y_t.

Con estas suposiciones, podemos definir la distribución conjunta usando solo tres distribuciones de probabilidad:

1. La distribución de $p(y_1)$.
2. La distribución $p(y_t \mid y_{t-1})$ de transición.
3. La distribución $p(x_t \mid y_t)$ de observación.

De esta forma obtenemos que la distribución conjunta es

$$p(Y, X) = p(y_1) \prod_{t=1}^{T} p(y_t \mid y_{t-1}) p(x_t \mid y_t) \tag{9.1}$$

9.3 Campos condicionales aleatorios de cadena lineal

Para motivar la introducción a los *CRFs* de cadena lineal, consideramos la distribución condicionada $p(y \mid x)$ que surge de la distribución $p(y, x)$ de un *HMM*.

En primer lugar, reescribimos la Ecuación 9.1 de forma que se facilite la generalización. Esto es

$$p(Y, X) = \frac{1}{Z} \prod_{t=1}^{T} \exp \left\{ \sum_{i,j \in S} \theta_{ij} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{y_{t-1} = j\}} + \sum_{i \in S} \sum_{o \in O} \mu_{oi} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{x_t = o\}} \right\} \tag{9.2}$$

donde $\theta = \{\theta_{ij}, \mu_{oi}\}$ son los parámetros reales de la distribución y Z es una constante de normalización.

El interés de este desarrollo es que podemos observar que la Ecuación 9.2 describe la clase de ecuaciones *HMMs* (9.1).

Podemos escribir la Ecuación 9.2 de forma más compacta introduciendo el concepto de *función característica*. Cada función característica tiene la forma

$$f_k(y_t, y_{t-1}, x_t). \tag{9.3}$$

Definiendo

$$f_{ij}(y, y', x) = \mathbf{1}_{y=i}\mathbf{1}_{y'=j} \tag{9.4}$$

para cada transición (i, j) y

$$f_{io}(y, y', x) = \mathbf{1}_{y=i}\mathbf{1}_{x=o} \tag{9.5}$$

para cada par estado-observación (i, o).

Entonces podemos escribir un HMM como:

$$p(Y, X) = \frac{1}{Z}\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y_t, y_{t-1}, x_t)\right\} \tag{9.6}$$

Nuevamente, la Ecuación 9.6 define exactamente el conjunto de ecuaciones *HMM* (9.1).

El último paso es escribir la distribución condicionada $p(Y \mid X)$ que surge de la Ecuación 9.6. Esto es:

$$p(Y \mid X) = \frac{p(Y, X)}{\sum_{Y'} p(Y', X)} = \frac{\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y_t, y_{t-1}, x_t)\right\}}{\sum_{Y'}\prod_{t=1}^{T}\exp\left\{\sum_{k=1}^{K}\theta_k f_k(y'_t, y'_{t-1}, x_t)\right\}} \tag{9.7}$$

La distribución condicional de la Ecuación 9.7 es un tipo particular de *CRF* de cadena lineal. Aquel que no incluye características solo de la palabra actual.

Otros *CRFs* de cadena lineal usan más características de la entrada, como prefijos y sufijos de lapalabra actual o la identidad de las palabras que la rodena.

Definition 9.1. *Sean Y, X dos vectores aleatorios, $\theta = \{\theta_k\} \in \mathbb{R}^K$ un vector de parámetros y $\mathcal{F} = \{f_k(y, y', x_t)\}_{k=1}^K$ un conjunto de funciones características reales. Entonces un CRF de cadena lineal es una distribución $p(Y \mid X)$ de la forma*

$$p(Y \mid X) = \frac{1}{Z(X)} \prod_{t=1}^{T} \exp\left\{ \sum_{k=1}^{K} \theta_k f_k(y'_t, y'_{t-1}, x_t) \right\} \tag{9.8}$$

donde $Z(X)$ es una función de normalización dependiente de la entrada

$$Z(X) = \sum_{Y} \prod_{t=1}^{T} \exp\left\{ \sum_{k=1}^{K} \theta_k f_k(y'_t, y'_{t-1}, x_t) \right\} \tag{9.9}$$

Para indicar en la definición de *CRF* de cadena lineal que cada función característica puede depender de las observaciones en cada paso de tiempo, hemos escrito al argumento de observación de f_k como el vector x_t. En nuestro problema particular se traduce en que el etiquetado de cada palabra va a depender de la entrada del resto de palabras, es decir, del contexto.

Por último, notar que la función de normalización $Z(X)$ realiza un sumatorio sobre todos los posibles estados de la secuencia, un número exponencialmente grande de términos. Este cálculo es inabarcable en el problema que tratamos y lo será en la mayoría de los casos, para ello introducimos el *algoritmo Viterbi*.

9.4 Algoritmo de Viterbi

El **algoritmo de Viterbi** es un algoritmo de programación dinámica que permite obtener la secuencia de estados ocultos más probable que produce una secuencia observada de sucesos, especialmente en el caso de *HMM*.

Soluciona el problema de cálculo inabarcable de la función de normalización $Z(X)$ en el caso de *CRF* de cadena lineal.

En primer lugar introducimos una notción que simplifica las recursiones hacia delante y hacia atrás. Un *HMM* puede ser descrito como $p(X, Y) = \prod_t \Psi_t(y_t, y_{t-1}, x_t)$ donde $Z = 1$ y los factores se definen como

$$\Psi_t(j, i, x) = p(y_t = j \mid y_{t-1} = i)p(x_t = x \mid y_t = j) \tag{9.10}$$

A continuación, introducimos el algoritmo hacia delante que se utiliza para calcular la probabilidad $p(X)$ de las observaciones.

La idea del algoritmo hacia delante y hacia atrás es primero reescribir la suma $p(x) = \sum_Y P(X, Y)$ utilizando la ley distributiva:

$$p(X) = \sum_Y \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, x_t) = \tag{9.11}$$

$$= \sum_{y_T} \sum_{y_{T-1}} \Psi_T(y_T, y_{T-1}, x_T) \sum_{y_{T-2}} \Psi_{T-1}(y_{T-1}, y_{T-2}, x_{T-1})... \tag{9.12}$$

Podemos observar que cada suma intermedia se reutiliza varias veces durante el cálculo de la suma externa, por tanto disminuir la carga de trabajo si guardamos las sumas internas.

Así, definimos un conjunto de *variables hacia delante* α_t, cada una de tamaño M donde M es el número de etados que representa las sumas intermedias. Se deinen de la siguiente forma:

$$\alpha_t(j) = \sum_{y_{<1\ldots t-1>}} \Psi_t(j, y_{t-1}, x_t) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \tag{9.13}$$

Los valores alfa se pueden calcular mediante la siguiente recursión

$$\alpha_t(j) = \sum_{i \in S} \Psi_t(j, i, x_t)\alpha_{t-1}(i), \tag{9.14}$$

inicializando $\alpha_1(j) = \Psi_1(j, y_0, x_1)$.

Se puede probar fácilmente por inducción que

$$p(X) = \sum_{y_T} \alpha_T(y_T) \tag{9.15}$$

sustituyendo repetidamente la recursión de la Ecuación9.14.

La recursión hacia atrás es exactamente igual, exceptuando la Ecuación 9.12, en la cual ponemos los sumatorios en orden inverso. Repitiendo el proceso obtenemos la definición de *variables hacia atrás*

$$\beta_t(i) = \sum_{y_{<t+1\ldots T>}} \prod_{t'=t+1}^{T} \Psi_{t'}(y_{t'}, y_{t'-1}x_{t'}), \tag{9.16}$$

y la recursión

$$\beta_t(i) = \sum_{j \in S} \Psi_{t+1}(j, i, x_{t+1})\beta_{t+1}(j) \tag{9.17}$$

inicializando $\beta_T(i) = 1$.

De forma análoga al caso hacia delante, podemos calcular $p(X)$ usando las variables hacia atrás de la siguiente forma

$$p(X) = \beta_0(y_0) = \sum_{y_1} \Psi_1(y_1, y_0, x_1)\beta_1(y_1) \tag{9.18}$$

Para calcular las distribuciones marginales $p(y_{t-1}, y_t \mid x)$, que serán neesarias para la estimación de parámetros, podemos combinar los resultados de las recursiones hacia delante y hacia atrás. En primer lugar, podemos escribir

$$p(y_{t-1}, y_t \mid X) = \frac{p(x \mid y_{t-1}, y_t)p(y_{t-1}, y_t)}{p(x)} \tag{9.19}$$

$$= \frac{p(x_{<1\ldots t-1>}, y_{t-1})p(y_t \mid y_{t-1})p(x_t \mid y_t)p(x_{<t+1\ldots T>} \mid y_t)}{p(x)} \tag{9.20}$$

$$= \frac{1}{p(x)}\alpha_{t-1}(y_{t-1})\Psi_t(y_t, y_{t-1}, x_t)\beta_t(y_t) \tag{9.21}$$

donde se ha utilizado en la segunda línea que $x_{<1\ldots t-1>}$ es independiente de $x_{<t+1\ldots T>}$. El factor $\frac{1}{p(x)}$ actúa como constante de normalización de la distribución. Podemos calcularla usando $p(x) = \beta_0(y_0)$ o $p(x) = \sum_{i \in S} \alpha_T(i)$.

Uniendo lo definido anteriormente, el *algoritmo hacia delante - hacia atrás* sería:

1. Calculamos α_t para cada t usando (9.14).
2. Calculamos β_t para cada t usando (9.17).
3. Obtenemos las distribuciones marginales usando (9.21).
4. Finalmente, calculamos la asignación más probable de la forma $y^* = \arg\max_y p(Y \mid X)$. Esto produce la *recursión de Viterbi*.

El análogo a las variables α en la versión hacia delante hacia detrás son

$$\delta_t(j) = \max_{y_{<1\ldots t-1}} \Psi_t(j, y_{t-1}, x_t) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \tag{9.22}$$

con su correspondiente recursión

$$\delta_t(j) = \max_{i \in S} \Psi_t(j, i, x_t)\delta_{t-1}(i) \tag{9.23}$$

Una vez que se han calculado las variables δ, la asignación de maximización puede calcularse con la recursión hacia atrás

$$y_T^* = \arg\max_{i \in S} \delta_T(i) \tag{9.24}$$

$$y_t^* = \arg \max_{i \in S} \Psi_t(y_{t+1}^*, i, x_{t+1}) \delta_t(i) \ \ \forall t < T \tag{9.25}$$

Las recursiones de δ_t y y_t^* de forma conjunta comprende el **algoritmo de Viterbi**.

DESARROLLO SOFTWARE Y ANÁLISIS DE RESULTADOS

En esta última parte de la memoria se explicará la solución software que se ha dado al problema de extracción de entidaes. En primer lugar se hará un estudio del problema cumpliendo los requisitos de la Ing. del Software para posteriormente proceder a la implementación de las soluciones planteadas. Se explicará el modelo inicial y por qué se ha elegido y se irá siguiendo una línea de mejoras para el resto de modelos.

Finalmente, se realizará un análisis comparativo de los modelos desarrollados concluyendo cuál es la mejor estructura para resolver el problema.

Capítulo 10

Análisis del problema

En este capítulo nos dedicaremos a realizar un análisis del problema que tratamos. En primer lugar se realizará un análisis de requisitos seguido de las herramientas que se utilizarán para su resolución.

10.1 Análisis de requisitos

El análisis de requisitos es un paso fundamental en el análisis del problema. Se describen en detalle los requisitos propios del sistema y el papel asignado al software. Podemos diferenciar dos tipos de requisitos: los funcionales y los no funcionales.

10.1.1 Requisitos funcionales

Se definen los requisitos funcionales como las funcionalidades finales que deberá ofrecer el proyecto de ingeniería.

- **[RF-1]: Entrenamiento del modelo**. Se podrán entrenar cada uno de los modelos desarrollados en unos datos concretos.
- **[RF-2]: Evaluación del modelo**. Se podrá evaluar el modelo una vez entrenado sobre otros datos de evaluación concretos.

10.1.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que especifican propiedades características del sistema, pero no intervienen en su funcionalidad.

- **[RNF-1]:** Los datos de entrenamiento del modelo estarán en el formato *XML* utilizado en *SemEval 2014*.
- **[RNF-2]:** La salida producida por el clasificador en la etapa de evaluación estarán en el mismo formato.
- **[RNF-3]: Usabilidad.** Para facilitar el proceso de entrenamiento se utilizará un archivo de texto para establecer los datos configurables de los modelos. Todos los campos de este fichero deberán estar rellenos. En caso contrario, el programa devolverá un mensaje de error.
- **[RNF-4] Escalabilidad** en el sentido de introducir nuevos modelos al sistema de forma fácil y sin cambiar la estructura.
- **[RNF-5]: Requisitos software**
  1. **Sistema operativo de desarrollo**: macOS High Sierra 10.13.4
  2. Python 3.6.3
  3. TensorFlow 1.9.0
  4. Keras 2.0.5
  5. **Editor de texto:** Sublime Text 2
  6. **Editor de diagramas:** Dia

10.2 Modelo de casos de uso

Los **casos de uso** representan fragmentos de funcionalidad del sistema, es decir, cada uno representa una función concreta del sistema en la que intervienen unos **actores** concretos. El primer paso para el modelado de los casos de uso es la identificación de los actores del sistema.

10.2.1 Actores del sistema

Se definen como **actores** del sistema a los elementos que interaccionan con el sistema. Dada la simplicidad funcional del proyecto habrá solo dos actores:

1. **[ACT_1] Ingeniero Informático experto en Ciencia de Datos:** encargado de configurar el modelo y de su posterior entrenamiento con unos datos concretos.
2. **[ACT_2] Usuario:** que realiza la evaluación y utilización del modelo ya entrenado.

10.2.2 Diagrama de casos de uso

En la mayoría de los casos, cada requisito funcional se correspondería con un caso de uso y por tanto con un diagrama de caso de uso independiente. Sin embargo, dada la simplicidad de las interacciones podemos representar los dos requisitos funcionales junto con sus actores en un mismo diagrama de caso de uso.

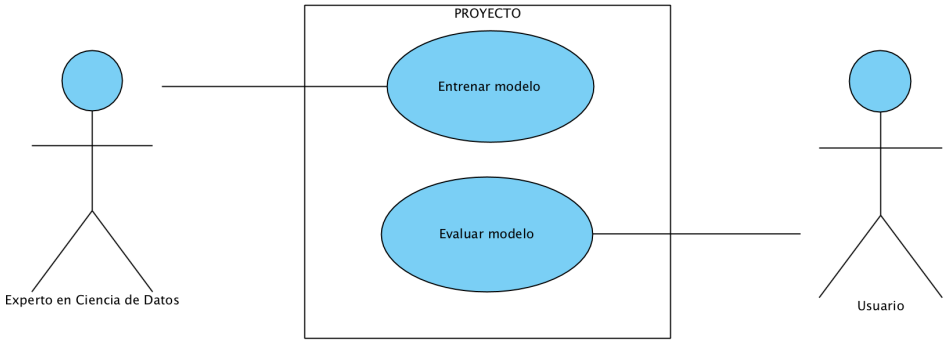


Figura 10.1: Diagrama de casos de uso (RF-1, RF-2).

En la Figura 10.1 se representa de forma esquematizada la interacción de los actores con el proyecto. Destacar que aunque se presenten como dos actores diferentes pueden ser la misma persona cumpliendo papeles diferenciados.

10.3 Modelo de dominio

Un **diagrama de modelo de dominio** es un paso de preparación para el diseño de clases. En el diagrama se muestran los conceptos significativos en el dominio del problema y sus relaciones. Se incluyen las restricciones a las cuales está sujeto el dominio.

En el dominio de nuestro problema encontramos tres conceptos clave:

1. **Clasificador:** modelo software que se encarga de etiquetar a cada palabra de cada opinión que recibe como entidadad o no. En nuestro caso se tratará de un modelo basado en redes neuronales.
2. **Opinión:** frase escrita en lenguaje natural que recibe el clasificador.
3. **Entidad:** palabra o conjunto de palabras que identifican un aspecto propio dentro de la opinión.

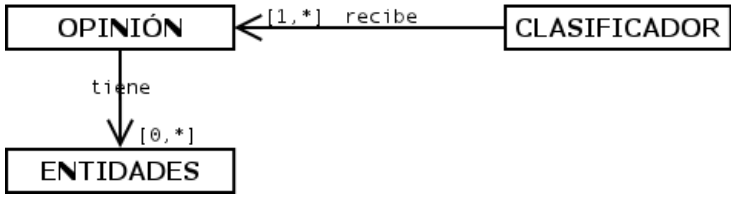


Figura 10.2: Diagrama de modelo de dominio.

En la Figura 10.2 vemos el dominio del problema. Nuestro objetivo es crear un clasificador que reciba un número de opiniones en las cuales debe encontrar las entidades presentes (puede no haber o puede haber varias).

10.4 Herramientas de desarrollo

Durante el proyecto se realiza el desarrollo de varios modelos basados en redes neuronales. Se han elegido herramientas de programación que facilitan el desarrollo de este tipo de modelos y que explicamos a continuación.

- **Lenguaje de Programación:** El lenguaje de programación elegido para el desarrollo del proyecto ha sido **Python 3.0** [1]. Python es un lenguaje fácil de usar, fácil de entender y versátil. Es un lenguaje de código abierto y multiplataforma, lo que permitió implementar el proyecto en OS X. Además, es un lenguaje maduro por lo que cuenta con varias bibiotecas para trabajar con todo tipo de datos de forma rápida y sencilla. Además de todas estas ventajas, el principal motivo por el cual se eligió como lenguaje para desarrollar el proyecto es por la existencia del **TensorFlow**.

- **TensorFlow**[2] es una biblioteca de software libre compatible con *Python 2.7-3.6* que se utiliza para realizar cálculos numéricos mediante el uso de diagramas de flujo. Se basa en una estructura de grafo donde los nodos representan operaciones matemáticas y las aristas tensores. En su origen fue creado para realizar investigaciones en el ámbito del aprendizaje automático y las redes neuronales profundas. Aunque a día de hoy su uso es muy generalizado sigue siendo adecuado para su primer cometido. Las principales ventajas de su uso son:

  - **Rendimiento:** La rapidez es una de sus principales ventajas dado que el rendimiento es fundamental en modelos tan pesados como las redes neuronales profundas. TensorFlow incluye *XLA*, un compilador de álgebra lineal que optimiza la velocidad de ejecución en procesadores y plataformas hardware integradas.

  - **Flexibilidad:** ofrece APIs de alto nivel para la simplificación del desarrollo y preparación de los modelos deseados. Para este proyecto se ha utilizado **Keras**, una famosa API diseñada para este cometido.

- **Keras**[3]**:** es una API [4] de alto nivel escrita en *Python* capaz de ejecutarse por encima de *TensorFlow*, *CNTK* o *Theano*. Su principal propósito es disminuir los tiempos necesarios para la implementación de modelos de Aprendizaje Automático. Para el proyecto hemos utilizado esta API con TensorFlow a los periodos de tiempo disponibles para el desarrollo

[1] `https://www.python.org/about/`

[2] `https://www.tensorflow.org/?hl=es`

[3] `https://keras.io/`

[4] API (Application Programming Interface), en español interfaz de programación de aplicaciones.

de dos modelos.

Para el manejo de datos, tanto lectura de datos para el entrenamiento de las redes como la producción de soluciones se han utilizado ficheros **XML (eXtensible Markup Languaje)**, en español "lenguaje marcado extensible". Se ha utilizado dado que es el que utilizan las bases de datos que utilizamos para el entrenamiento y por su facilidad de lectura utilizando *Python*.

Otro objetivo, es la comparación de los modelos implementados. Para ello hemos utilizado la hoja de cálculo de **OpenOffice** para obtener los cálculos estadísticos de las repeticiones de los experimentos. Además, con este mismo programa se han realizado las gráficas que se utilizarán en el apartado de la comparación de resultados.

Para el diseño de las gráficas de la parte de Fundamentos Teóricos se ha utilizado *RStudio* dada la facilidad para dibujar gráficas de funciones en *R* por la existencia de varias bibliotecas para ello.

Para el diseño de los esquemas *UML* que se utilizarán en la Sección de diseño (Sección 11) se ha utilizado el software **Dia**[5]. Se ha elegido este software por su carácter libre, característica poco común en los softwares que presentan similar funcionalidad.

Para el desarrollo de toda la memoria y las representaciones de grafos dirigidos para la explicación de las redes neuronales se han utilizado:

- **LaTeX:** es un sistema de preparación de textos marcado, es decir, el usuario utiliza etiquetas para indicar qué estructura quiere crear. Es cómodo y mantiene el estilo a lo largo del documento. Su utilización es simplifica con el uso de entornos integrados como *TexStudio*.
- **TexStudio** es un entorno integrado de escritura para crear documentos *LaTeX*. Hace su uso más cómodo dado que proporciona atajos de teclado, sugerencias, etcétera. Además es incorpora un compilador que devuelve el correspondiente documento pdf.

Finalmente, para el manejo de versiones y coordinación con los tutores se ha utilizado la plataforma *GitHub*.

[5] `http://dia-installer.de/index.html.en`

Capítulo 11

Diseño

En este capítulo se desarrollará el diseño de la solución implementada siguiendo los principios de la Ingeniería del Software. Para ello realizaremos un diagrama de clases haciendo uso del Patrón de Diseño Software Estrategia [5].

11.1 Diagrama de clases

En esta sección veremos la estructura de clases que se ha seguido durante el desarrollo del proyecto. Como veremos es importante la usabilidad y escalabilidad del sistema resultante.

Dado que uno de los principales objetivos del proyecto es la comparación de diferentes modelos aplicados a un mismo problema, es muy importante la escalabilidad horizontal. Es decir, la estructura debe permitir la introducción de nuevos modelos sin cambiar el código de los anteriores. Para el diseño de la estructura cumpliendo tales requisitos se ha utilizado el Patrón de diseño Estrategia.

11.1.1 Patrón de diseño Estrategia

El patrón de diseño para desarrollo software Estrategia (en inglés *Strategy*) es un patrón de comportamiento pues determina cómo debe realizarse el intercambio de mensajes entre objetos. Una de sus aportaciones es que permite mantener diferentes algoritmos entre los cuales el usuario puede elegir dinámicamente.

Usando este patrón de diseño se ha creado una clase principal **Model_RRNN** de la cual heredarán el resto de modelos. Esta clase se encarga de la funcionalidad general del modelo:

- Lectura de datos de entrenamiento y test.
- Lectura de los *embeddings* a utilizar.
- Preparación de los datos en formatos adecuados para las entradas de las redes.
- Cálculo de las funciones de error
- Escritura del fichero salida con la solución calculada por el modelo

De esta forma, cada modelo heredará de esta clase teniendo que definir la funcionalidad específica de cada modelo.

Como en esta práctica hemos realizado dos modelos diferentes, la estructura de clases de forma generalizada que hemos utilizado sería la que se muestra en la Figura 11.1.

Cada uno de los modelos implementa de forma particular:

- Configuración del modelo según el tipo de red que sea y los parámetros configurables que se hayan establecido.
- Entrenamiento del modelo con los datos de entrenamiento.
- Ajuste del modelo al entrenamiento.
- Predicción de los datos de test que realiza el modelo ya entrenado.
- Cargar el modelo de un modelo previamente guardado.

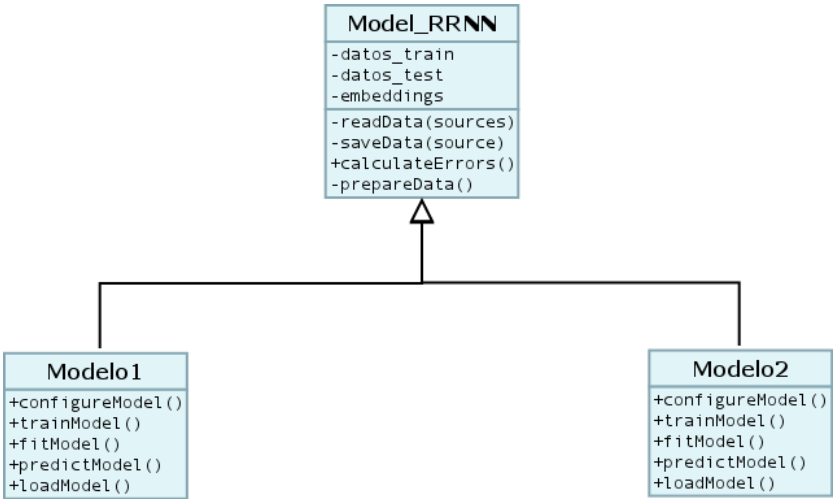


Figura 11.1: Jerarquía de clases en los modelos implementados.

Resaltar que los métodos específicos de los modelos, aunque se comporten de forma diferente dependiendo del modelo concreto se llaman igual. Esto ayuda a la simplificación de código dado que solo debemos especificar el tipo de modelo a la hora de la creación, el resto del código será reutilizable.

11.1.2 Lector de datos

Para la lectura y adecuación al formato necesitado para el problema se ha creado una clase **XMLData** que se encarga de leer el fichero *XML* con la estructura preestablecida y formar una base de datos con formato **IOB**.

El formato **IOB (In - Out - Begin)** trata de una tokenización de una oración etiquetando cada uno de los tokens con una etiqueta I, O o B en función de su papel en una entidad:

1. **B** cuando representa el principio (*begin*) de una entidad.
2. **I** cuando se encuentra dentro (*in*) de una entidad y no es la primera palabra.
3. **O** cuando no pertenece (*out*) a ninguna entidad.

Por ejemplo, la frase

Este fin de semana viajo a Nueva York.

tendría la siguiente tokenización IOB

(Este, O) (fin,B) (de,I) (semana,I) (viajo,O) (a,O) (Nueva,B) (York,I).

donde podremos observar que en la oración completa hay un total de dos entidades compuestas por tres y dos palabras respectivamente.

La preparación de los datos de esta forma se ha realizado utilizando el módulo para manejo de datos *XML* de *Python* **xml.etree.ElementTree**.

11.1.3 Arquitectura software

En cuanto a la separación de la funcionalidad total de los modelos en la dos funcionalidades comentadas en la Sección 10.1.1 y de la escalabilidad anteriormente comentada se ha utilizado un **Modelo Vista Controlador (MVC)**.

El **MVC** es una patrón de arquitectura software que divide el diseño en tres partes:

- **Modelo:** parte más importante del proyecto pues es donde se encuentra toda la funcionalidad. Se corresponde con la clase *Model_RRNN* y las clases que heredan de ella.
- **Vista:** Capa software que tiene la funcionalidad de introducir los datos y devolver la solución. En este caso es la terminal junto con el formato de ficheros anteriormente comentado.
- **Controlador:** El modelo y la vista deben ser totalmente independientes. El controlador se encarga de relacionar la vista (datos introducidos y de salida) con el modelo (procesamiento de estos datos). Dada la simplicidad de la interacción en el modelo desarrollado no se precisa de controlador entre el modelo y la vista.

Para ello, tenemos dos nuevas clases **Trainer** y **Predictor** que se encargan de llamar a los métodos propios de cada modelo para llevar a cabo cada funcionalidad.

Así, la estructura general del proyecto sería la siguiente:

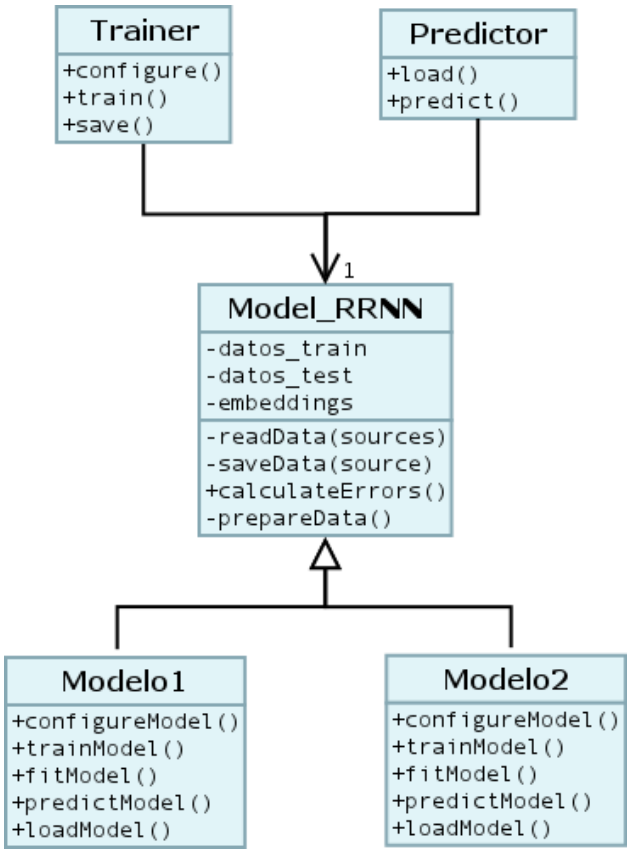


Figura 11.2: Diagrama de clases de la parte funcional del modelo.

Capítulo 12

Implementación y Experimentación

Una vez diseñado el modelo se procede a su implementación. En este capítulo abarcaremos los detalles de la implementación de cada una de las partes en las que se compone el proyecto.

12.1 Lectura y procesamiento de datos

Esta sección es, quizás, la más novedosa dada la naturaleza del problema. En este proyecto estamos tratando con opiniones escritas en lenguaje natural. Sin embargo, los modelos que utilizamos, como ya hemos adelantado, son redes neuronales que no trabajan con carácteres alfanuméricos y menos aún con oraciones completas.

Por ello, necesitamos un preprocesamiento de los datos para convertirlos en datos entendibles por las redes neuronales al mismo tiempo que siguen manteniendo las cualidades de la oración como el significado y el contexto.

12.1.1 Tokenización

El primer paso a la hora de tratar con oraciones completas que obtenemos de un fichero de datos es *tokenizar* la oración, esto es, dividirla en los tokens (componente léxico con significado propio) que la componen. En la práctica esto se traduce a transformar la oración que tenemos almacenada en un vector de palabras.

Para la *tokenización* de frases se ha utilizado el método *word_tokenize* de **NLTK**[1] **(Natural Language ToolKit)**, módulo para el procesamiento del lenguaje natural en Python.

12.1.2 Relleno o truncamiento

Una vez se tiene la frase *tokenizada* hay que homogeneizar las longitudes de las diferentes frases que componen los conjuntos de datos. Esto se debe a las limitaciones de los modelos que estamos utilizando.

Para ello, establecemos una longitud de oración fija y rellenamos las frases más cortas y truncamos las que superen esta longitud.

En nuestro caso se ha elegido tamaño máximo de oración 65.

12.1.3 *Word Embeddings*

Uno de los principales hándicaps del procesamiento de lenguaje natural está en el tipo de datos utilizados. El conjunto de datos se compone de conjuntos de oraciones ya *tokenizadas* en palabras. Sin embargo, la mayoría de los modelos de Aprendizaje Automático aceptan como entrada números reales.

Durante el desarrollo del PLN se han utilizado diferentes medidas para caracterizar a las palabras. Una de las caracterizaciones más utilizadas es el *TF-IDF*[16] (del inglés *Term frequency - Inverse document frecuency*) que asigna a cada palabra un número real que representa su frecuencia de aparición. De este modo, el valor será mayor cuanto más veces aparezca en el documento dotándola de mayor importancia.

[1] `https://www.nltk.org/`

Dado que los algoritmos de *Deep Learning* precisan de un mayor número de características para su buen funcionamiento, se han utilizado los *word embeddings* como representación para cada palabra.

Un ***word embedding*** es un vector de longitud prefijada que toma valores reales y que representa el significado de cada palabra. Este tipo de representación es la más utilizada en el estado del arte dado que ha proporcionado buenos resultados.

A la hora de utilizar este tipo de representación podemos:

1. Utilizar *word embeddings* preentrenados. Contienen un conjunto de palabras (vocabulario) a las cuales asignan un vector determinado. A la hora de utilizar este tipo de *word embeddings* se suelen mantener constantes durante todo el aprendizaje.

2. Utilizar *word embeddings* inicializados aleatoriamente y que se ajustan por el modelo utilizado como el resto de parámetros.

Los *word embeddings* preentrenados han mostrado un buen funcionamiento en la literatura por lo que para este proyecto se utilizan los *word embeddings* preentrenados de propósito general de *Glove: Gloval Vectors for Word Representation - Stanford NLP Group*[2]. En este conjunto preentrenado cada *word embedding* se corresponde con un vector de 300 características. Dada las dimensiones del fichero, con 2.03GB, y las limitaciones *hardware* utilizaremos sólo los 500000 primeros.

Tanto por el uso de *word embeddings* preentrenados formando un vocabulario ya fijado al conjunto en el cual se ajustaron, como por las limitaciones de memoria que nos han hecho considerar sólo un conjunto de ellos, utilizamos dos tipos de *word embeddings* especiales:

1. **Vector de relleno:** asignado a aquellos tokens de relleno en las frases que eran más cortas que la longitud prefijada de oración. Será un vector de longitud 300 rellenado con ceros.

2. **Vector desconocido:** asignado a aquellos tokens que no se encuentren identificados con ningún *embedding* entre los preentrenados considerados. Será un vector de longitud 300 rellenado de forma aleatoria.

[2] `https://nlp.stanford.edu/projects/glove/`

Una vez asignado a cada token el *word embedding* correspondiente tendríamos como representación de cada frase una matriz $A \in \mathbb{R}^{65\times300}$ que podrá ser la entrada de una red neuronal.

12.2 Características de lo datos utilizados

Para las pruebas de los modelos implementados se han utilizado los conjuntos de entrenamiento y test de comentarios sobre portátiles de *SemEval-2014: Semantic Evaluation Exercises* [3] al igual que el artículo [12].

El conjunto se compone de:

1. 3045 opiniones de entrenamiento.
2. 800 opiniones de test.

12.3 Etiquetado

Dado que se trata de un problema de clasificación multietiqueta, necesitamos el conjunto de datos de entrenamiento etiquetado al mismo tiempo que debemos producir una salida con el etiquetamiento de los datos de test.

Dado que la clasificación será IOB, anteriormente comentada, tendríamos tres tipos de etiquetas. Si a estas tres etiquetas le añadimos una etiqueta de relleno para aquellos tokens que se hayan añadido en el relleno de oraciones tenemos que el problema resultante sería un problema de clasificación con cuatro etiquetas. Dado que es más fácil trabajar con números que con caracteres alfanuméricos la identificación de etiquetas que se ha utilizado a lo largo de toda la implementación es:

- **3 - I** *in*
- **1 - O** *out*
- **2 - B** *begin*
- **0 - P** (*padding*)

[3] `http://alt.qcri.org/semeval2014/task4/`

12.4 Modelos implementados

Como ya se ha comentado, se han implementado dos modelos diferentes:

1. Uno basado en una CNN, al cual haremos una ligera mejora.
2. Uno basado en una biLSTM.

Ambos modelos se han implementado utilizando la API de alto nivel Keras para TensorFlow. Esta API facilita las funciones de entrenamiento (*compile*), ajuste (*fit*) y predicción (*predict*).

Para la creación de cada uno se han especificado las capas deseadas utilizando métodos del módulo *layers* de Keras.

La estructura de los modelos se especificarán en el siguiente capítulo junto con los resultados obtenidos con cada uno de ellos.

Con respecto al etiquetado secuencial explicado en el Capítulo 9 se ha implementado utilizando funciones de *CRF* ofrecidas por TensorFlow. Para ello, se ha reimplementado la función de pérdida de las redes neuronales haciendo uso de las funciones que dicho módulo ofrece [7] [13].

A la hora de calcular la secuencia que minimiza la función de pérdida redefinida para asignarlas como etiquetas finales se ha utilizado el algoritmo Viterbi. Para lo que también se han utilizado funciones predefinidas de TensorFlow.

Podemos concluir esta sección corroborando el buen criterio a la hora de elegir Python y TensorFlow a la hora de la implementación dado el gran repertorio de módulos para aprendizaje automático y manejo de datos que ofrece.

12.5 Medidas de error consideradas

Dada la naturaleza del problema se pueden considerar varias formas de medir el error producido por un modelo diferentes.

1. Por su naturaleza de clasificación multiclase se pueden considerar las medidas ***macro-averages***. Esto es, consideramos la media de la precisión y del *recall* de cada una de las subclases de la siguiente forma:

$$PRE_{macro} = \frac{PRE_B + PREC_I + PREC_O + PREC_{Padding}}{4}$$

$$RECALL_{macro} = \frac{RECALL_B + RECALL_I + RECALL_O + RECALL_{Padding}}{4}$$

La ventaja de esta medida es que se puede calcular haciendo uso de funciones del módulo para aprendizaje automático de TensorFlow *scikit*[4].

El problema que estas medidas suponen es que al ser la media de las precisiones por etiqueta, se puede discriminar a una clase en concreto haciendo que las otras suban considerablemente.

Además, en nuestro problema concreto pueden ser más interesantes otras medidas que destaquen el número de etiquetas referentes a pertenencia a entidades que se han acertado.

2. **Etiquetado parcial:** es el etiquetado que se considera en el artículo [12]. Trata de medir la precisión y el *recall* a la hora de detectar la pertenencia de palabras a entidades.

   - La precisión mide la cantidad de etiquetados B o I acertadas. Esto es, de entre las palabras etiquetadas como B o I, cuales están bien etiquetadas.
   - El *recall* mide la cantidad de etiquetas B o I encontradas. Esto es, de entre todas las palabras cuya etiqueta es como B o I, cuáles han sido etiquetadas correctamente.

3. **Etiquetado total:** considera el etiquetado correcto si se ha encontrado la entidad completa.

   - La precisión mide la cantidad de entidades bien etiquetadas. Esto es, de entre todos los conjuntos de palabras etiquetados como entidad, cuáles se corresponden con una entidad de verdad. En el caso en el que se encuentre solo parte de una entidad o se etiquete como entidad a un conjunto más grande de palabras de las correctas, se considera como erróneo.

[4] `http://scikit-learn.org/stable/`

- El *recall* mide la cantidad de entidades encontradas completamente entre el número de entidades total del problema.

En la práctica, esta medida será la más útil pues el tener entidades totalmente etiquetadas será necesario para proceder a la asignación de polaridad por entidades que sería el siguiente paso natural de este campo del PLN. De esta forma, este tipo de etiquetado proporciona una medida más rigurosa del comportamiento del modelo.

En las tres formas de medida de error se ha considerado, además de la precisión y el *recall* una medida que resume ambos valores, el *F1-score* o F1 que se calcula mediante la siguiente fórmula:

$$F1 = 2 \times \frac{PRE \times RECALL}{PRE + RECALL}$$

Capítulo 13

Modelos y Resultados

En este capítulo se desarrollan en detalle los modelos implementados a la vez que se expilan los motivos de las estructuras elegidas. También se realizará un análisis de resultados comparativo entre los modelos implementados.

13.1 Arquitecturas implementadas

Durante el desarrollo del proyecto se han implementado tres arquitecturas diferentes:

- Una red neuronal convolucional (CNN) reimplementación de la que se explica en el artículo [12].
- Se propone una mejora al primer modelo ampliando el tamaño de las convoluciones (CNN-4-6)
- Una vez estudiadas las limitaciones de dichas arquitecturas, se propone un modelo basado en *biLSTM*

Para la experimentación y comparación de resultados en los diferentes modelos se han fijado los parámetros comunes:

1. Tamaño de *batch* de 16, dado el reducido tamaño del conjunto de entrenamiento.

2. El número de *epochs* o iteraciones se ha fijado a 30 siguiendo las instrucciones del artículo [12].

3. Se ha repetido 20 veces cada experimento.

13.2 Primer modelo: CNN

13.2.1 Estructura

El primer modelo se trata de un algoritmo basado en redes neuronales convolucionales. Aunque no sea lo más común en este tipo de problemas elegir una red convolucional por su naturaleza secuencial, se ha elegido esta estructura basándonos en el artículo [12].

La estructura se trata de una reimplementación de la arquitectura explicada en el artículo. Consta de las siguientes partes:

1. Dos capas de convolución. Las dimensiones de las convoluciones serían: mapas de características de tamaño 100 y 50 y núcleos de tamaño 2 y 3 respectivamente. En ambas convoluciones utilizamos como función de activación la tangente hiperbólica (ver Sección 6.1.3).

2. En cada convolución elegimos el *stride* a 1 para no modificar la dimensión de la oración (queremos una etiqueta por palabra).

3. Tras cada convolución se aplica una capa de *pooling*(ver Sección 7.2), en concreto *max-pooling* de tamaño 2.

4. Tras cada *pooling* aplicamos regularización basada en *dropout* con parámetro de regularización 0,5.

5. Finalmente, aplicamos una capa densa reduciendo al número de clases que contemplamos (cuatro). De esta forma, obtendríamos un vector de 65×4 para cada oración. Cada vector de 4 dimensiones representa la probabilidad de pertenencia a cada una de las clases.

Podemos apreciar el esquema general que obtenemos usando *TensorBoard*[1].

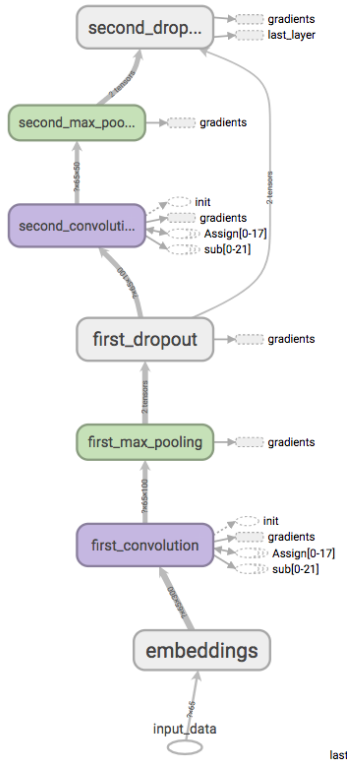


Figura 13.1: Estructura simplificada de la red del modelo CNN.

Además, TensorFlow también tiene la opción de mostrar el *summary* o resumen de la red. Esta herramienta nos ofrece la estructura en capas junto con las dimensiones tras aplicar cada capa y el número de parámetros.

La dimensión *None* se corresponde con el tamaño del *batch*. Comprobamos que la primera capa produce una entrada de $(None, 65)$ correspondiéndose a *batch* frases de 65 palabras cada una. Tras la capa de *embeddings* aumentan en una dimensión asignando a cada palabra un vector de tamaño 300. Tras la primera convolución disminuímos esta última dimensión a 100 (especificado por el mapa de características) y se mantienen las dimensiones invariantes

[1]Herramienta que combinada con TensorFlow nos ofrece en tiempo real gráficas del error y del accuracy al mismo tiempo que la estructura completa de la red

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_data (InputLayer)      (None, 65)                0
_________________________________________________________________
embeddings (Embedding)       (None, 65, 300)           150000300
_________________________________________________________________
first_convolution (Conv1D)   (None, 65, 100)           120100
_________________________________________________________________
first_max_pooling (MaxPoolin (None, 65, 100)           0
_________________________________________________________________
first_dropout (Dropout)      (None, 65, 100)           0
_________________________________________________________________
second_convolution (Conv1D)  (None, 65, 50)            30050
_________________________________________________________________
second_max_pooling (MaxPooli (None, 65, 50)            0
_________________________________________________________________
second_dropout (Dropout)     (None, 65, 50)            0
_________________________________________________________________
last_layer (Dense)           (None, 65, 4)             204
_________________________________________________________________
```

Figura 13.2: Estructura por capas de la red.

hasta la siguiente capa de convolución, donde análogamente se disminuye a 50. Por último tras la capa densa obtenemos como dimensión de salida la deseada $(None, 65, 4)$

13.2.2 Medidas del error

En la Tabla 13.1 tenemos la media y la varianza de los resultados obtenidos en las 20 ejecuciones del modelo y los resultados obtenidos en el artículo [12].

		CNN		Artículo
		Media	**Varianza**	**Media**
Macro	**Precisión**	0.92132	2.07×10^{-5}	-
	Recall	0.92426	2.1×10^{-5}	-
	F1	0.92279	2.04×10^{-5}	-
Etiq. total	**Precisión**	0.78600	2.87×10^{-4}	-
	Recall	0.64176	5.3×10^{-4}	-
	F1	0.70625	2.05×10^{-4}	-
Etiq. parcial	**Precisión**	0.89372	8.3×10^{-4}	-
	Recall	0.72280	1.5×10^{-3}	-
	F1	0.79922	1.9×10^{-4}	0.7732

Tabla 13.1: Resultados ejecución del primer modelo y comparativa.

En cuanto a la reimplementación del artículo [12] vemos que aunque el resultado sea muy similar se ha superado en un 3,3 %. Una diferencia tan pequeña se puede deber al uso de diferentes *word embeddings* (aunque ambos de propósito general).

Con respecto a los resultados obtenidos por el primer modelo, la reimplementación, observamos que en general la varianza de los datos es muy pequeña lo cual nos indica que los resultados no están muy dispersos. Por tanto, las conclusiones que saquemos de estas ejecuciones podrán servir para una buena estimación del comportamiento del modelo.

La medida de error que mejores resultados nos proporciona es la macro, sin embargo es la que menos fiabilidad aporta por el desbalanceo de clases en nuestro problema concreto (abundancia de etiquetas O y Padding frente a entidades (I-B)).

Por otro lado, el etiquetado total es bastante más bajo que el etiquetado parcial (entorno a 12 % menos). Esto nos confirma la importancia de esta medida pues una parte considerable de las palabras que etiquetaba parcialmente no las está etiquetando bien de forma completa.

Realizamos a continuación un análisis en el que estudiamos los motivos de estos resultados y las posibles soluciones. l

13.2.3 Análisis

En esta sección se estudia el comportamiento del modelo en oraciones concretas para analizar sus limitaciones y así realizar mejoras al modelo enfocadas a un objetivo concreto.

Las conclusiones que se han sacado del análisis son las siguientes:

1. En la mayoría de los casos clasifica bien las entidades de una o dos palabras.

2. En la mayoría de los casos en los que etiqueta una palabra parcialmente etiqueta bien el principio (B) pero falla en el número de palabras que la componen. En la inmensa mayoría de estos casos el error se produce al etiquetar como entidad menos palabras de las que realmente la componen. Esto es, en el caso de entidades de tres o cuatro palabras el

modelo identifica solo las dos primeras palabras. Un ejemplo concreto en el que se ha detectado el fallo es en el siguiente:

[1] (One, O) (night,O) (I,O) (turned,O) (the,O) (freaking,O) (thing,O) (off,O) (after,O) (using,O) (it,O) (, ,O) (the,O) (next,O) (day,O) (I,O) (turn,O) (it,O) (on,O) (, ,O) (no,O) ***(GUI,B)*** *(, , O)* ***(screen,B)*** *(all,O) (dark,O) (, ,)* ***(power,B) (light,I)*** *(steady,O) (, ,)* ***(hard,B) (drive, I) (light, I)*** *(steady, O) (and,O) (not,O) (flashing,O) (as,O) (it,O) (usually,O) (does,O).*

donde el modelo ha etiquetado de la siguiente forma:

[1] (One, O) (night,O) (I,O) (turned,O) (the,O) (freaking,O) (thing,O) (off,O) (after,O) (using,O) (it,O) (, ,O) (the,O) (next,O) (day,O) (I,O) (turn,O) (it,O) (on,O) (, ,O) (no,O) ***(GUI,B)*** *(, , O)* ***(screen,B)*** *(all,O) (dark,O) (, ,)* ***(power,B) (light,I)*** *(steady,O) (, ,)* ***(hard,B) (drive, I) (light, O)*** *(steady, O) (and,O) (not,O) (flashing,O) (as,O) (it,O) (usually,O) (does,O).*

3. En las oraciones en las que aparecen varias entidades, el modelo tiende a etiquetar de forma incorrecta las entidades del final de la oración.

4. En varias ocasiones el modelo asigna la etiqueta B a palabras que no forman parte de ninguna entidad disminuyendo de esta forma la precisión.

En función del análisis realizado, se cree que el problema reside en el desconocimiento que tiene una palabra concreta del resto de palabras que la rodean. Gracias a la convolución, el vector que la caracteriza tendrá información de las palabras de un pequeño vecindario pero no del resto de la oración. Esto explica el porqué del mal etiquetado de entidades formadas por más de dos palabras. Superar estas limitaciones será el primer objetivo planteado.

13.3 Segundo modelo: CNN-4-6

13.3.1 Estructura

El segundo modelo surge de la idea de intentar mejorar las limitaciones del primer modelo. Dado que el principal problema era la limitación de contexto para este segundo modelo se propone ampliar el tamaño de filtro de las convoluciones para así ampliar el marco de contexto.

La estructura de este segundo modelo será la misma pero con tamaño de filtro de ambas convoluciones 4 y 6 respectivamente.

13.3.2 Medidas de error

En la tabla 13.2 tenemos la media y varianza de los resultados obtenidos en las 20 ejecuciones del modelo.

		Media	Varianza
Macro	**Precisión**	0.92099	9.2×10^{-6}
	Recall	0.92684	8.4×10^{-6}
	F1	0.92390	8.3×10^{-6}
Etiquetado total	**Precisión**	0.80542	9.6×10^{-4}
	Recall	0.65517	9.7×10^{-4}
	F1	0.72243	1.6×10^{-4}
Etiquetado parcial	**Precisión**	0.93637	1.7×10^{-5}
	Recall	0.70783	8.1×10^{-5}
	F1	0.80612	3.7×10^{-5}

Tabla 13.2: Resultados ejecución del segundo modelo.

En este caso, la varianza de los datos es incluso más pequeña por lo que las conclusiones serán de igual modo fiables.

En comparación con el modelo anterior, observamos que en todos los etiquetados contemplados aumenta el *F1*. Notar que de las dos formas de etiquetado (parcial y total) contempladas se mejora mucho en precisión. Este hecho nos indica la disminución del error en el caso del etiquetado de palabras como entidades, proporcionando resultados más fiables y precisos.

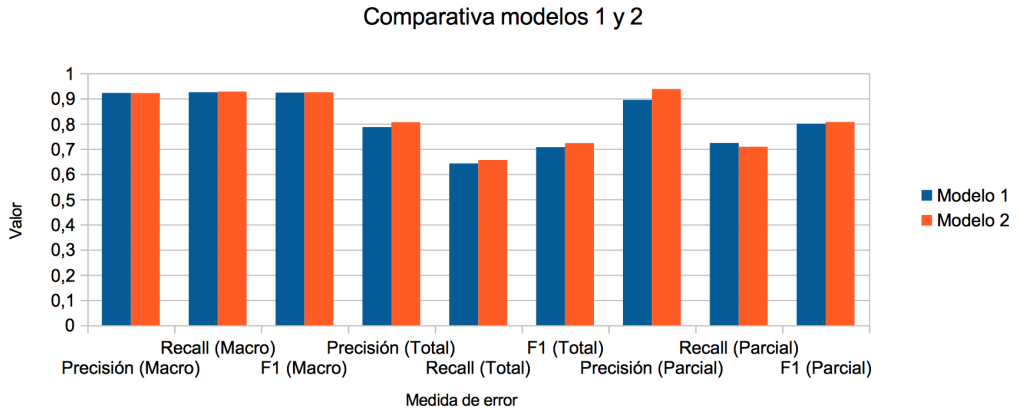


Figura 13.3: Gráfica comparativa de ambos modelos.

Aunque en general la diferencia de resultados en el *F1* es pequeña, la mayor diferencia se obtiene en el etiquetado total en el cual sí se mejoran ambas medidas (precisión y *recall*). Dado que los resultados obtenidos con este etiquetado son más rigurosos, podemos concluir que se ha superado al primer modelo ampliando el conocimiento de las palabras de alrededor, aunque aún se encuentra muy sesgado.

13.3.3 Análisis

En esta sección, comapraremos los resultados obtenidos con el modelo anterior al mismo tiempo que encontraremos las limitaciones del nuevo modelo, pudiendo concluir que son las limitaciones del tipo de estructura que hemos elegido.

Las observaciones que se han realizado son la siguientes:

1. Con respecto al ejemplo considerado en el apartado anterior:

   [1] (One, O) (night,O) (I,O) (turned,O) (the,O) (freaking,O) (thing,O) (off,O) (after,O) (using,O) (it,O) (, ,O) (the,O) (next,O) (day,O) (I,O) (turn,O) (it,O) (on,O) (, ,O) (no,O) ***(GUI,B)*** *(, , O)* ***(screen,B)*** *(all,O) (dark,O) (, ,)* ***(power,B) (light,I)*** *(steady,O) (, ,)* ***(hard,B) (drive, I) (light, I)*** *(steady, O) (and,O) (not,O) (flashing,O) (as,O) (it,O) (usually,O) (does,O).*

el nuevo modelo ha realizado el siguiente etiquetado

[1] (One, O) (night,O) (I,O) (turned,O) (the,O) (freaking,O) (thing,O) (off,O) (after,O) (using,O) (it,O) (, ,O) (the,O) (next,O) (day,O) (I,O) (turn,O) (it,O) (on,O) (, ,O) (no,O) ***(GUI,O)*** *(, , O)* ***(screen,B)*** *(all,O) (dark,O) (, ,)* ***(power,B)*** ***(light,I)*** *(steady,O) (, ,)* ***(hard,B)*** ***(drive, I)*** ***(light, I)*** *(steady, O) (and,O) (not,O) (flashing,O) (as,O) (it,O) (usually,O) (does,O).*

Vemos que aunque el modelo ha etiquetado correctamente la entidad de tres palabras, ha fallado en etiquetar una entidad de una sola palabra. Este hecho se ha detectado en más de una oración, explicando, en parte, por qué aún detectándose entidades más largas no aumenta considerablemente el *recall*.

2. Además, aún detectando la mayoría de las entidades de tres palabras, sigue teniendo limitaciones en cuanto al número de palabras en una entidad que detecta. Un ejemplo sería:

   [1] (So,O) (, , O) (I,O) (paid, O) (a, O) (visit, O) (to, O) ***(LG,B)*** ***(notebook,I)*** ***(service, I)*** ***(center, I)*** *(at,O) (Alexandra,O) (Road,O) (, , O) (hoping, O) (they, O) (can, O) (make, O) (the, O)* ***(hinge, B)*** *(tighter, O) (. , O)*

   donde el nuevo modelo ha realizado el siguiente etiquetado

   [1] (So,O) (, , O) (I,O) (paid, O) (a, O) (visit, O) (to, O) ***(LG,B)*** ***(notebook,I)*** ***(service, I)*** ***(center, O)*** *(at,O) (Alexandra,O) (Road,O) (, , O) (hoping, O) (they, O) (can, O) (make, O) (the, O)* ***(hinge, O)*** *(tighter, O) (. , O)*

   Vemos que en la entidad de cuatro palabras detecta solo las tres primeras al mismo tiempo que no detecta la entidad de una palabra. Esto ocurre dado que aunque la ventana de contexto en este caso sea mayor, sigue estando limitado por la poca ocurrencia de este tipo de entidades en los conjuntos de entrenamiento y el contexto sesgado que tiene cada palabra.

3. En cuanto al problema del modelo anterior de asignar B a palabras que no forman parte de ninguna entidad se disminuye en este modelo. Posible motivo por el cual aumenta la precisión del etiquetado tanto

parcial como total.

4. En cuanto al problema de detectar peor las entidades que se encuentran al final de las oraciones en aquellas que tengan muchas entidades se sigue manteniendo. Esto ocurre dado que en la mayoría de las oraciones hay una o dos entidades.

En conclusión, las limitaciones del modelo se resumen en la falta de contexto pues está muy limitado al tamaño de filtro. Esto limita al modelo a aprender casos más concretos como entidades más complejas u oraciones en las que aparezcan más de dos entidades. El problema de la falta de contexto se debe al mal comportamiento de las redes neuronales convolucionales en el modelado de secuencias.

Por tanto, la dirección a seguir en la busca de una mejora será un cambio en la estructura de la red. En la línea en la que se desarrolla el estado del arte para este tipo de problemas, la siguiente estructura elegida será una red neuronal recurrente (en concreto una *biLSTM*).

13.4 Tercer modelo: biLSTM

13.4.1 Estructura

Una vez estudiadas las limitaciones de las redes neuronales convolucionales en este tipo de problemas, se propone utilizar una arquitectura más adecuada dada la naturaleza del problema, las redes neuronales recurrentes [3] [10].

Como ya se ha abordado en el Capítulo 8, las redes neuronales recurrentes suelen ser más adecuadas para el modelado de secuencias.

El carácter recurrente de estas redes proporciona resultados dependientes del contexto. En la práctica, la arquitectura más adecuada de estas redes sería una red con memoria a corto-largo plazo ($LSTM$) . Estas redes garantizan que a la hora de etiquetar una palabra concreta se tenga información de las palabras anteriores. Sin embargo, lo ideal sería tener información en ambos sentidos. Para ello, se propone como estructura una *biLSTM* de 256 capas ocultas que consiste básicamente en aplicar una capa recurrente en cada uno de los sentidos.

Por tanto, la estructura que se propone sería de la siguiente forma

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_data (InputLayer)      (None, 65)                0
_________________________________________________________________
embedding_1 (Embedding)      (None, 65, 300)           150000300
_________________________________________________________________
bidirectional_1 (Bidirection (None, 65, 512)           1140736
_________________________________________________________________
last_layer (Dense)           (None, 65, 4)             2052
=================================================================
```

Figura 13.4: Estructura de la red propuesta.

Observamos en la figura 13.5 que la estructura del nuevo modelo es muy básica, sin embargo, veremos que proporciona buenos resultados.

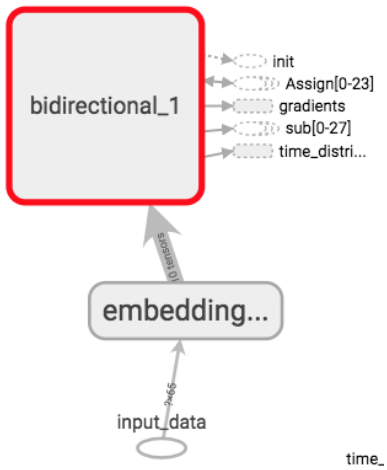


Figura 13.5: Estructura de la red biLSTM.

13.4.2 Medidas de error

En la tabla 13.3 tenemos la media y varianza de los resultados obtenidos en las 20 ejecuciones del modelo.

La varianza de los datos sigue en la línea de los modelos anteriores, por lo que podemos realizar la misma conclusión.

		Media	Varianza
Macro	**Precisión**	0.92910	3.1×10^{-6}
	Recall	0.93373	4.2×10^{-6}
	F1	0.93141	3.5×10^{-6}
Etiquetado total	**Precisión**	0.85985	5.6×10^{-5}
	Recall	0.68033	9.9×10^{-5}
	F1	0.75565	5.2×10^{-5}
Etiquetado parcial	**Precisión**	0.93987	5.8×10^{-5}
	Recall	0.71689	7.3×10^{-5}
	F1	0.81407	1.1×10^{-5}

Tabla 13.3: Resultados ejecución del segundo modelo.

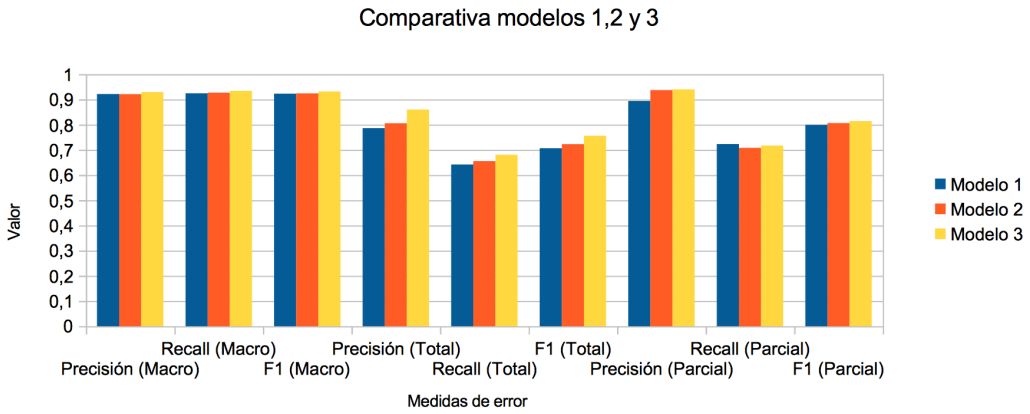


Figura 13.6: Comparativa de los tres modelos desarrollados.

En comparación con los modelos anteriores, se ha obtenido mejor F1 para los tres etiquetados. En concreto, para el etiquetado total se ha obtenido una mejora de un 4,4 % con respecto al anterior modelo (CNN-4-6) siendo la mayor de las mejoras.

Analizando la precisión y el *recall* notamos que aunque se ha producido mejoras en ambas medidas, la mayor parte de la mejora se ha producido en precisión. Esto nos indica, que aunque el nuevo modelo no encuentre muchas más entidades, la precisión de la clasificación es mayor, lo cual aporta fiabilidad al modelo.

13.4.3 Análisis

En esta sección se estudiará el comportamiento comparándolo con los modelos anteriores. Además, se intentarán encontrar las limitaciones del modelo de cara un posible trabajo futuro.

Las observaciones que se han realizado son las siguientes:

1. Con respecto a la opinión en la que los otros modelos fallaron algún etiquetado

   [1] (One, O) (night,O) (I,O) (turned,O) (the,O) (freaking,O) (thing,O) (off,O) (after,O) (using,O) (it,O) (, ,O) (the,O) (next,O) (day,O) (I,O) (turn,O) (it,O) (on,O) (, ,O) (no,O) ***(GUI,B)*** *(, , O)* ***(screen,B)*** *(all,O) (dark,O) (, ,)* ***(power,B) (light,I)*** *(steady,O) (, ,)* ***(hard,B) (drive, I) (light, I)*** *(steady, O) (and,O) (not,O) (flashing,O) (as,O) (it,O) (usually,O) (does,O).*

   el nuevo modelo ha encontrado todas las etiquetas

   [1] (One, O) (night,O) (I,O) (turned,O) (the,O) (freaking,O) (thing,O) (off,O) (after,O) (using,O) (it,O) (, ,O) (the,O) (next,O) (day,O) (I,O) (turn,O) (it,O) (on,O) (, ,O) (no,O) ***(GUI,B)*** *(, , O)* ***(screen,B)*** *(all,O) (dark,O) (, ,)* ***(power,B) (light,I)*** *(steady,O) (, ,)* ***(hard,B) (drive, I) (light, I)*** *(steady, O) (and,O) (not,O) (flashing,O) (as,O) (it,O) (usually,O) (does,O).*

2. El modelo sigue representando limitaciones a la hora de reconocer entidades de varias palabras. Por ejemplo, en el caso considerado en el modelo anterior tenemos que comete el mismo error:

   [1] (So,O) (, , O) (I,O) (paid, O) (a, O) (visit, O) (to, O) ***(LG,B) (notebook,I) (service, I) (center, I)*** *(at,O) (Alexandra,O) (Road,O) (, , O) (hoping, O) (they, O) (can, O) (make, O) (the, O)* ***(hinge, B)*** *(tighter, O) (. , O)*

   el nuevo modelo ha realizado el siguiente etiquetado

   [1] (So,O) (, , O) (I,O) (paid, O) (a, O) (visit, O) (to, O) ***(LG,B) (notebook,I) (service, I) (center, O)*** *(at,O) (Alexandra,O) (Road,O) (, , O) (hoping, O) (they, O) (can, O) (make, O) (the, O)* ***(hinge, B)*** *(tighter, O) (. , O)*

Esto se debe a que, aunque en este modelo se tenga en cuenta el contexto de toda la oración, las oraciones en las que aparecen entidades de tantas palabras son excasas en el conjunto de enternamiento, dificultando el aprendizaje para este tipo de casos.

3. El número de casos en los que el modelo etiqueta como entidad (B-I) una palabra que no pertenece a ninguna entidad es casi inexistente. La pérdida de precisión en este caso en el etiquetado parcial se da cuando etiqueta como I a una palabra que debería tener la etiqueta B o viceversa, siendo un error "menos grave" que el anterior.

4. La mayoría de la pérdida de precisión en el caso del etiquetado total se da en entidades con más de 3 palabras o en la confusión de etiquetas B e I (menos común). Por tanto, en la práctica es mayor la precisión y el número de entidades encontradas (*recall*).

13.5 Comparativa de modelos

En esta sección, comentamos brevemente gráficas y tablas representativas de la mejora que se ha producido a lo largo del desarrollo de los diferentes modelos.

		Artículo	CNN	CNN-4-6	biLSTM
Macro	**Precisión**	-	0.92132	0.92099	0.92910
	Recall	-	0.92426	0.92684	0.93373
	F1	-	0.92279	0.92390	0.93141
Total	**Precisión**	-	0.78600	0.80542	0.85985
	Recall	-	0.64176	0.65517	0.68033
	F1	-	0.70625	0.72243	0.75565
Parcial	**Precisión**	-	0.89372	0.93637	0.93987
	Recall	-	0.72280	0.70783	0.71689
	F1	0.7732	0.79372	0.80612	0.81407

Tabla 13.4: Tabla comparativa de los modelos.

En la Tabla 13.4 podemos ver de forma resumida lo que hemos comentado en las secciones anteriores. En general, el modelo basado en redes neuronales

recurrentes obtiene los mejores resultados para casi todas las medidas de error comentadas e independientemente del etiquetado elegido.

Sorprendentemente, el modelo inicial es el que mejor resultado obtiene en *recall* para el etiquetado parcial. Esto significa que es el modelo que más palabras pertenecientes a entidades ha encontrado. Sin embargo, hay que analizar a qué coste se ha alcanzado el *recall* más alto pues presenta un 5 % menos de precisión con respecto al mejor modelo. Esto nos indica que el primer modelo tiene mayor predisposición a etiquetar palabras como partes de una entidad.

Basándonos en el hecho de que considerando el etiquetado total no ocurre este fenómeno, podemos concluir que la mejor solución contemplada es el tercer modelo propuesto.

Sin embargo, no todo son ventajas. Si analizamos los tiempos necesarios para la ejecución de una prueba obtenemos los siguientes resultados:

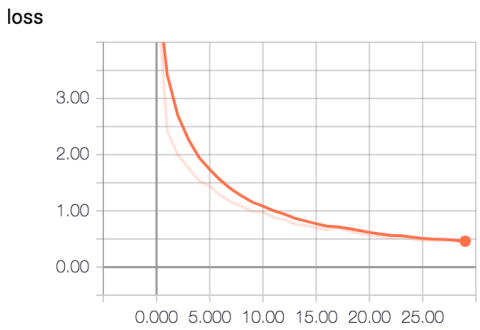
	CNN	**CNN-4-6**	**biLSTM**
Tiempo medio (s)	69.3	184.5	1815.2

Tabla 13.5: Tabla de tiempos medios de entrenamiento.

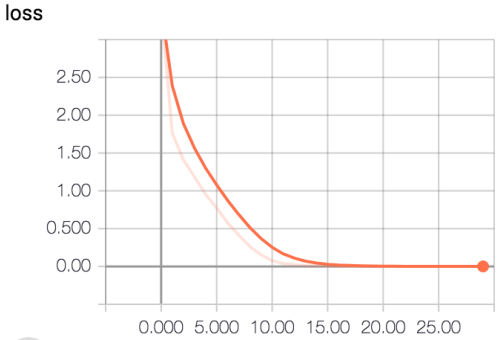
Con cada mejora que hemos conseguido, se ha ido incrementando el tiempo medio empleado en una ejecución. Para el entrenamiento de la red recurrente se ha necesitado 26 veces más tiempo que para el primer modelo convolucional. Esto se traduce en algo más de media hora por repetición. Teniendo en cuenta que hemos realizado 20 ejecuciones para obtener datos medios y así analizar su funcionamiento, para las pruebas de la red recurrente se han necesitado más de 10h mientras que las del primer modelo se realizaron en poco más de media hora.

Aunque sea una diferencia considerable, aún son tiempos abarcables por lo que no suponen un problema. Destacar que los tiempos son de entrenamiento, a la hora de utilizar un modelo ya entrenado no se produciría tal inconveniente.

Analizamos ahora, haciendo uso de la herramienta TensorBoard otra vez, la progresión de la función de coste definida para el mejor modelo de redes convolucionales (CNN-4-6) y la biLSTM:



(a) CNN-4-6



(b) BiLSTM

El modelo biLSTM, además de comenzar en un error más bajo (poco por encima de 3) comienza con un descenso mucho más acentuado, terminando en una asíntota horizontal en el 0 a partir de la época 15. Sin embargo, la red neuronal convolucional apenas baja de 0.5 en la función de coste transcurridas las 30 épocas. Este hecho aporta más evidencia al buen funcionamiento de redes en problemas de esta naturaleza.

A modo de conclusión de esta capítulo, resaltar la mejora que hemos conseguido con respecto al primer modelo considerado, que recordemos era una reimplementación del artículo [12]. Además, se ha visto la importancia de la arquitectura recurrente en el modelado secuencial.

Capítulo 14

Conclusiones y trabajo futuro

En este capítulo se resumen las conclusiones principales del proyecto. Algunas de ellas darán lugar a posibles líneas de desarrollo de trabajo futuro.

- Analizando los resultados obtenidos con el primero modelo (CNN) y trabajando en mejorarlos (CNN-4-6) se ha llegado a la conclusión de que el principal problema era la arquitectura elegida. Ambas redes neuronales convolucionales propuestas han presentado limitaciones debido a la falta de conocimiento del contexto que se tenía a la hora del etiquetado.

- Con una arquitectura sencilla de biLSTM se han conseguido buenos resultados. Ha quedado reflejado el buen comportamiento de este tipo de arquitecturas en problemas de modelado de secuencias, como se desarrolló de forma teórica anteriormente.

- Se ha conseguido superar los resultados del artículo [12] con ambos modelos propuestos. Con el modelo basado en biLSTM se ha llegado a mejorar la precisión en un 5 % y el F1 en un 2,5 % siendo mejoras considerables.

- Como vía futura de este trabajo se propone el desarrollo de un sistema software basado en biLSTM competente con el estado del arte. Dado el buen comportamiento que ha presentado una arquitectura sencilla con respecto al artículo tomado como referencia, se propone partir del último modelo desarrollado realizando ciertas mejoras como:

- Arquitectura más compleja contemplando más capas.
- Combinar con capas de convolución para combinar modelado secuencial con conocimiento espacial (convolución).
- Utilizar *modelos de atención* en ciertas partes de la oración.
- Estudio del comportamiento de la red en más conjuntos del estado del arte.

Bibliografía

[1] Daniel Bestard Delgado. "¿cómo se explica el crecimiento del big data en la última década? ¿qué retos nos ha planteado esta ciencia?". *Evaluación de Impacto*, 2013.

[2] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman and Hall/CRC, 2nd ed. edition, 2004.

[3] Guillaume Genthial. Sequence tagging with tensorflow, 2015.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] Craig Larman. *UML y Patrones*. PEARSON EDUCACION, 2 edition, 2003.

[6] Bing Liu. *Sentiment Analysis and Opinion Mining*. Morgan and Claypool Publishers, 2012.

[7] Eugenio Martínez Cámara, Vared Shwartz, Iryna Gurevych, and Ido Dagan. Neural disambiguation of causal lexical markers based on context. In *Proceedings of the 12th International Conference on Computational Semantics (IWCS 2017)*, September 2017.

[8] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[9] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proc. of ICML*, pages 807–814, 2010.

[10] Christopher Olah. Understanding lstm networks, 2015.

[11] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis, 2008.

[12] Soujanya Poria, Erik Cambria, and Alexander Gelbukh. Aspect extraction for opinion mining with a deep convolutional neural network. *Knowledge-Based System*, 2016.

[13] Nils Reimers and Iryna Gurevych. Reporting Score Distributions Makes a Difference: Performance Study of LSTM-networks for Sequence Tagging. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 338–348, Copenhagen, Denmark, 09 2017.

[14] Sheldon M. Ross. *Introduction to Probability Models*. Academic Press, San Diego, CA, USA, sixth edition, 1997.

[15] Charles Sutton and Andrew McCallum. *An Introduction to Conditional Random Fields*. now, 2014.

[16] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting tf-idf term weights as making relevance decisions. *ACM Trans. Inf. Syst.*, 26(3):13:1–13:37, June 2008.