# Búsquedas de respuestas con Deep Learning

Trabajo de fin de grado
Grado en Ingeniería Informática, Facultad de Informática
Universidad Complutense de Madrid
Director: Alberto Díaz Esteban
Codirector: Antonio F. García Sevilla

Ignacio Terriza Díez y Daniel Reyes Parrilla

Curso 2017/2018


UNIVERSIDAD COMPLUTENSE
MADRID

# Agradecimientos

A nuestro director y a Antonio F. G. Sevilla por su predisposición a ayudarnos durante la realización de un TFG tan complejo.

A nuestras familias por el apoyo y ayuda económica para poder hacer la carrera.

A Andrew Ng. por sus cursos de machine lerning.

A Google por ofrecernos gratuitamente sus GPUs para la ejecución de nuestro programa.

A Roberto Díaz Badra por explicarnos como funciona el Google Colaboratory.

# Resumen

Esta memoria explica, en pimer lugar, como funcionan sistemas de pregunta y respuesta existentes en la actualidad. En concreto, se estudian diversos sistemas relacionados con la medicina, debido que en un primer momento, en este trabajo se pretendía implementar un chat médico. A partir de conocer como funcionan los sistemas existentes y tener claro lo que se quiere conseguir, se sigue explicando los fundamentos técnico-teóricos más importantes del Deep Learning de cara a la implementación final del sistema de preguntas y respuestas. En concreto se hace mucho énfasis en los fundamentos del Machine Learning, los tipos diferentes de Redes Neuronales Recurrentes y como usarlas, y la arquitectura Encoder-Decoder. En el siguiente capítulo se explica la teoría del procesamiento del lenguaje natural. Este apartado de la memoria se explica en conjunto con Spacy, para darle también un enfoque más práctico. Al finalizar se explican los diferentes métodos de representación del lenguaje en vectores numéricos. Después empezará la segunda parte, y la más importante del TFG, que es como se han aplicado estos conceptos teóricos al sistema de preguntas y respuestas final. Se empieza por explicar lo que hace el código de manera secuencial y muy abstracta evadiéndose en todo momento del código en sí, para ahondar en por qué funciona de una forma más conceptual y teórica. A continuación, se hace mención a cada uno de los experimentos a los que se sometió el sistema y se comentan los resulados. Finalmente se procede a dar una opinión personal de por qué se han obtenido tales resultados y las soluciones que se le darían, haciendo hincapié en que se tienen que demostrar todos de manera empírica.

Palabras clave: Machine Learning, Deep Learning, Sequence to sequence, Keras, Spacy, Encoder-Decoder, RNN.

# Abstract

This report explains, first of all, how question answering systems currently exist. Specifically, various systems related to medicine are studied, because at first, in this work was intended to implement a medical chat. From knowing how the existing systems work and being aware about what we wanted to achieve, this document continues explaining the most important technical-theoretical foundations of Deep Learning for the final implementation of the question answering system. In particular, much emphasis is placed on the fundamentals of Machine Learning, the different types of Recurrent Neural Networks and how to use them, and the Encoder-Decoder architecture. The theory of natural language processing is explained in the next chapter. This section of the report is explained in conjunction with Spacy, to also give you a more practical approach. At the end, the different methods of representing the language in numerical vectors are explained. Then the second and most important part of the TFG begins, which explains how these theoretical concepts have been applied to the final question answering system. It begins by explaining what the code does in a very abstract and sequential manner, evading the code itself at all times, to focus into why it works in a more conceptual and theoretical way. Next, each of the experiments to which the system was underwent is mentioned and the results are discussed. Finally we proceed to give a personal opinion of why such results have been obtained and the solutions that would be given to it, emphasizing that they have to be demonstrated empirically.

Key words: Machine Learning, Deep Learning, Sequence to sequence, Keras, Spacy, Encoder-Decoder, RNN.

# Índice general

# Índice de figuras

# Capítulo 1

# Introducción

## 1.1. Motivación del proyecto

Uno de los temas que más preocupa al ser humano es la salud, por eso hemos decidido hacer un sistema de respuestas médicas. Desde los comienzos de la humanidad ha habido personas encargadas de velar por nuestro bienestar, poniendo a nuestra disposición remedios (más o menos efectivos) para nuestras enfermedades y dolencias. En la antigüedad un médico o curandero conocía remedios para un reducido número de enfermedades, pero hoy en día el estado de la medicina es bastante avanzado, y cada día avanza más rápido, lo que requiere de unos profesionales con amplios conocimientos sobre una vasta cantidad de temas. Por este motivo cada vez es más difícil que los profesionales de la salud sean capaces de recordar en todo momento toda la información necesaria para realizar un diagnóstico óptimo. Al mismo tiempo, muchas veces los pacientes creemos que estamos enfermos cuando en realidad no lo estamos, debido en parte a nuestro desconocimiento de la materia. Con todo lo anterior en mente planteamos el siguiente Trabajo de Fin de Grado, buscando dar solución a ambos problemas hemos investigado si es posible mediante Deep Learning, una de las ramas más avanzadas de la Inteligencia Artificial, construir un chatbot capaz de orientar tanto al médico como al paciente en temas de la salud. Concretamente hemos tratado de construir un sistema de preguntas y respuestas en el ámbito médico, para disuadir al paciente de ocupar la consulta en caso de no ser necesario, al igual que para facilitar al médico el acceso a la información en un modo más humano (preguntas y respuestas).

Existen buscadores como por ejemplo, PUBMED o GOPUBMED, que resuelven de manera parcial este problema. El inconveniente es que se centran en un limitado rango de recursos específicos cuando muchas de estas fuentes a menudo necesitan ser combinadas y estudiadas. Por ejemplo, bases de datos y ontologías especializadas en medicamentos. La mayoría de las veces recogen textos relevantes para un tema o información estructurada, la cual implica mucho trabajo por parte del usuario para obtener las respuestas adecuadas. En contraste con esto último, nosotros, mediante nuestro sistema de QA (Question-Answering, o Pregunta-Respuesta) queremos producir esas respuestas de una manera mucho más directa. Estos métodos, hoy en día, requieren de mas investigación para

alcanzar un nivel de eficiencia y eficacia aceptable por los expertos en medicina. De esta manera nuestro sistema de respuestas será un granito de arena, que sin perder de vista que queremos cumplir con el objetivo propuesto, servirá de investigación para nuevos sistemas más capaces. Aunque comenzamos con la idea de implementar un chatbot completo, pronto nos dimos cuenta de que era un proyecto demasiado ambicioso y decidimos acotar el problema hacia un sistema de preguntas y respuestas, componente principal de cualquier chatbot.

## 1.2. Objetivos

Al comenzar el proyecto tan solo teníamos claro que queríamos hacer Deep Learning. Para centrar el desarrollo de nuestro sistema, usamos BioASQ, una serie de desafíos en el ámbito de la bioinformática, concretamente preguntas y respuestas, debiendo utilizarse la técnica de extracción de información de textos. Así pues nos inscribimos en la tarea "BioASQ 5b", especializada en preguntas y respuestas, de donde obtuvimos el data set que hemos utilizado, aunque al final no nos dió tiempo a presentar nuestros resultados al concurso.

La tarea de BioASQ ha sido atacada desde varios enfoques, con modelos extractivos (sacando subcadenas de los textos de entrada [14]) o basados en búsquedas [19], pero ninguno buscando una relación directa entre preguntas y respuestas a través de una red neuronal.

### 1.2.1. Diseño de un chatbot

La primera pregunta que hay que contestar es qué es para nosotros un chatbot inteligente. Para nosotros un chatbot inteligente es aquel que mantiene una conversación lo más parecido a como lo haría un ser humano y además acierta con una probabilidad muy alta las preguntas que se le hacen. Dependiendo del cometido de nuestro chatbot, éste debe saber contestar tanto preguntas sobre el pasado, cosas que se saben o ya han ocurrido, como poder pronosticar lo que va a suceder a partir de unas premisas.

Para poder hacer que el bot converse como un ser humano es muy importante estar al tanto de las necesidades del usuario. Por ejemplo si sabemos del usuario que trabaja como un obrero el chatbot debería evitar ofrecerle medicamentos que produzcan somnolencia como efecto secundario pues esto tendría un efecto negativo en su rendimiento laboral. Aun así un AI (Artificial Intelligence o Inteligencia Artificial) chatbot está basado en la capacidad humana de autoaprendizaje eficiente, como por ejemplo aprender de las conversaciones que ya ha tenido para mejorar su rendimiento. De este modo es necesario que el chatbot comprenda y emita el lenguaje natural. Hay herramientas como **IBM Watson, Api.ai, and Wit.ai** para incorporar la habilidad del lenguaje natural al bot.

Los algoritmos de ML (Machine Learning) junto con supervisores humanos pueden hacer que el chatbot aprenda. Para asegurar que el AI chatbot se convierte en un buen alumno usaremos redes neuronales y deep learning. Con el aprendizaje el bot es capaz de reconocer patrones en los datos que recibe y responder

a las peticiones del usuario de la manera más apropiada.

Para poder responder a cada petición del usuario requiere seguir un proceso de tres etapas:

- **Comprender el ambiente que nos rodea**: Necesitamos entender lo que dice el usuario para poder usar dicho conocimiento como prerrequisito para poder encontrar la solución que el cliente desea.
- **Pensar**: El bot debe transformar la información recibida en un formato que logre entender y guardar en su base de conocimiento. Usaremos esta base de conocimiento que se irá actualizando en todo momento para tomar las decisiones que sean oportunas. Si tomamos los casos de Siri y Google Now podemos observar que esta base de conocimiento les ayuda a aprender más rápido, indentificar la información relevante y proveer una respuesta que satisfaga las demandas del usuario. El **análisis predictivo (predictive analytics)** usando ML puede hacer que el bot prediga las consultas que le vaya a realizar el usuario.
- **Actuar**: Ahora que ya sabemos con precisión que quiere el usuario debemos darle la información precisa expresada de la mejor manera para que la comprenda.

También es importante saber los límites de la conversación. Hay normalmente dos posibilidades, la primera donde la conversación no llega a ningún fin y podremos hablar de cualquier tema y otras en las que tengas un objetivo claro, por ejemplo en una consulta médica el dominio será acotado.

Para generar respuestas existen dos tipos de modelos:

- **Retrieval-based**: hay respuestas guardadas y según el contexto se decide cuál usar.
- **Generative-based**: el bot es capaz de generar nuevas respuestas cada vez.

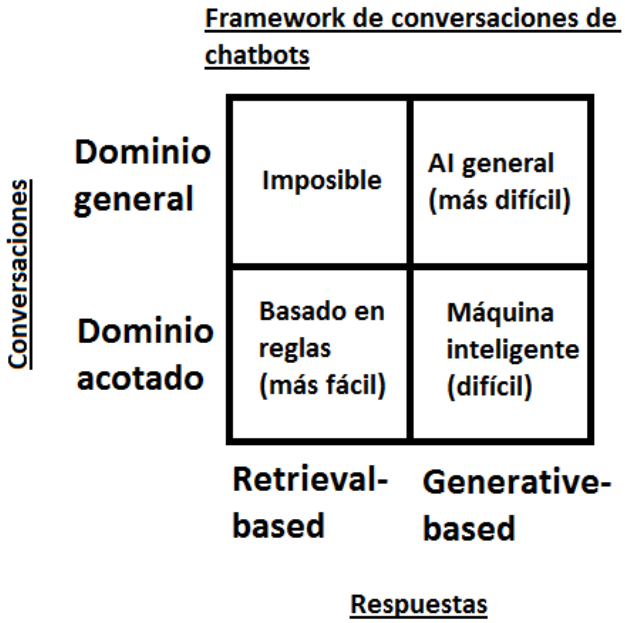


Figura 1.1: Clasificación de distintos modelos de chatbot según el dominio y el modo de generar las respuestas

De esta manera los objetivos más importantes para construir un buen IA chatbot son:

- Integrar el contexto.
- Generar respuestas coherentes.
- Elegir y construir bien el modelo.
- Entrenarlo de modo que tenga una buena base de conocimiento.

Para transformar el lenguaje en acciones y contextos debemos usar NLP (Natural language processing) con deep learning.

### 1.2.2. Estado del arte de chatbots

Hay varios ejemplos de chatbots ya disponibles en la red. Hemos investigado y según la opinión pública los mejores son entre otros: Mitsuku, Rose, Right Click, Poncho, Insomo bot, Dr A.I, Melody by baidu. Las más importantes de analizar para nuestro objetivo son Dr A.I y Melody by baidu puesto que son dos chatbots médicos.

Dr A.I : Entre sus funcionalidades están tener una conversación empática con el cliente sobre sus síntomas y salud general para conocer el contexto. Más tarde y con ayuda de la Inteligencia Artificial y Machine Learning el Dr A.I. inmediatamente traduce los sintomas en posibles explicaciones médicas que estarán

personalizadas según la edad, género, posibles condiciones externas, medicamentos, y otros aspectos relevantes de la salud del paciente. Por último Dr A.I. da un tratamiento médico adecuado con la mejora del paciente, después de todo el proceso te da la oportunidad de hablar con un médico real.

Melody by baidu: Este chatbot al contrario que el anterior está pensado para que lo use el médico y no el paciente. Proveé al médico de información relevante para asistir al paciente con recomendaciones útiles y posibles tratamientos. Melody incorpora Deep learning avanzado y NLP. Además los doctores usan Melody para que los propios pacientes lo tengan en casa para poder obtener respuestas rápidas sobre sus problemas de salud.

## 1.3. Estructura del documento

Tras esta introducción, el capítulo 3 describe el funcionamiento de las distintas técnicas utilizadas (Machine Learning, Redes Neuronales y Deep Learning) tanto de aprendizaje automático como de análisis de texto. En el capítulo 4 explicaremos la extracción de propiedades del lenguaje y una de las librerías de NLP de software libre más potentes: Spacy. En el capítulo 5 comentaremos la elección del lenguaje informático y entorno de ejecución. En el siguiente capítulo (6) explicaremos nuestra arquitectura de generación de respuestas en el ámbito biomédico, se describe el proceso de adaptación de las distintas técnicas al problema concreto y todo lo que hemos tenido que aprender. El capítulo 7 está dedicado a los experimentos realizados y el análisis de resultados. Finalmente, el capítulo 8 consiste en las conclusiones y un planteamiento del trabajo futuro.

# Capítulo 2

# Introduction

## 2.1. Project motivation

One of the issues that most concerns the human being is health, that is why we have decided to make a system of medical responses. Since the beginning of humanity there have been people in charge of watching over our well-being, putting at our disposal remedies (more or less effective) for our illnesses and ailments. In ancient times a doctor or healer knew remedies for a small number of diseases, but today the state of medicine is quite advanced, and every day advances faster, which requires professionals with extensive knowledge about a vast amount of topics. For this reason it is becoming more difficult for health professionals to be able to remember at all times all the information necessary to make an optimal diagnosis. At the same time, often patients believe that they are sick when in fact they are not, due in part to their ignorance of the matter. With all the above in mind we propose the following End of Degree Project, seeking to solve both problems we have investigated whether it is possible through Deep Learning, one of the most advanced branches of Artificial Intelligence, to build a chatbot able to guide both the doctor and the patient in health issues. Specifically we have tried to build a question answering system in the medical field, to dissuade the patient from occupying the consultation in case of not being necessary, as well as to facilitate the doctor access to information in a more human way (questions and answers).

There are search engines such as PUBMED or GOPUBMED, which solve this problem partially. The issue is that they focus on a limited range of specific resources when many of these sources often need to be combined and studied. For example, databases and ontologies specialized in medicines. Most of the time they collect relevant texts for a topic or structured information, which involves a lot of work on the part of the user to obtain the appropriate answers. In contrast to the latter, we, through our system of QA (Question-Answering), want to produce those answers in a much more direct way. These methods, nowadays, require more research to reach a level of efficiency and efficacy acceptable to medical experts. In this way, our response system will be, without losing sight of the fact that we want to meet the proposed objective, a contribution to help research for new and more capable systems. Although we started with the idea

of implementing a complete chatbot, we soon realized that it was too ambitious and we decided to narrow the problem to a question and answer system, the main component of any chatbot.

## 2.2. Goals

At the beginning of the project, we only knew that we wanted to do Deep Learning. Our directors told us about BioASQ, a series of challenges in the field of bioinformatics, specifically questions and answers, where the technique of extracting information from texts must be used. So we signed up for the task "BioASQ 5b", specialized in questions and answers, from where we obtained the data set we used, although in the end we did not have time to present our results to the contest.

The task of BioASQ has been attacked from several approaches, with extractive models (removing substrings from the entry texts [14]) or based on searches [19], but none seeking a direct relationship between questions and answers through of a neural network.

### 2.2.1. Design of a chatbot

The first question to answer is what is an intelligent chatbot for us. For us, an intelligent chatbot is the one that maintains a conversation as close as a human being would, and also answers correctly the questions that are asked with a very high probability. Depending on the role of our chatbot, it must know how to answer questions about the past, things that are known or have already happened, and how to predict what is going to happen from a few premises.

In order to make the bot converse as a human being it is very important to be aware of the user's needs. For example, if we know of the user who works as a labourer, the chatbot should avoid offering drugs that produce drowsiness as a secondary effect, as this would have a negative effect on his work performance. Even so, an AI chatbot is based on the human capacity of efficient self-learning, such as learning from the conversations it has already had to improve its performance. In this way it is necessary that the chatbot understand and emit the natural language. There are tools like **IBM Watson, Api.ai, and Wit.ai** to incorporate the natural language ability to the bot.

The algorithms of ML (Machine Learning) together with human supervisors can make the chatbot learn. To make sure that the AI chatbot becomes a good student we will use neural networks and deep learning. With the learning process, the bot is able to recognize patterns in the data it receives and respond to user requests in the most appropriate way.

In order to respond to each user's request, a three-step process must be followed:

- **Understand the environment that surrounds us**: We need to understand what the user says in order to use that knowledge as a prerequisite

to find the solution that the client wants.

- **To think**: The bot must transform the information received in a format that it can understand and save in its knowledge base. We will use this knowledge base that will be updated at all times to make the decisions that are appropriate. If we take the cases of Siri and Google Now we can see that this knowledge base helps them to learn faster, identify the relevant information and provide a response that meets the user's demands. The **predictive analytics** using ML can make the bot predict the queries that the user will make.
- **To act**: Now that we know precisely what the user wants, we must give him the precise information expressed in the best way so that he understands it.

It is also important to know the limits of the conversation. There are usually two possibilities, the first where the conversation does not go to any end and we can talk about any topic and others in which you have a clear goal, for example in a medical consultation the domain will be bounded.

To generate answers there are two types of models:

- **Retrieval-based**: there are saved answers and depending on the context, it is decided which one to use.
- **Generative-based**: The bot is capable of generating new responses every time.

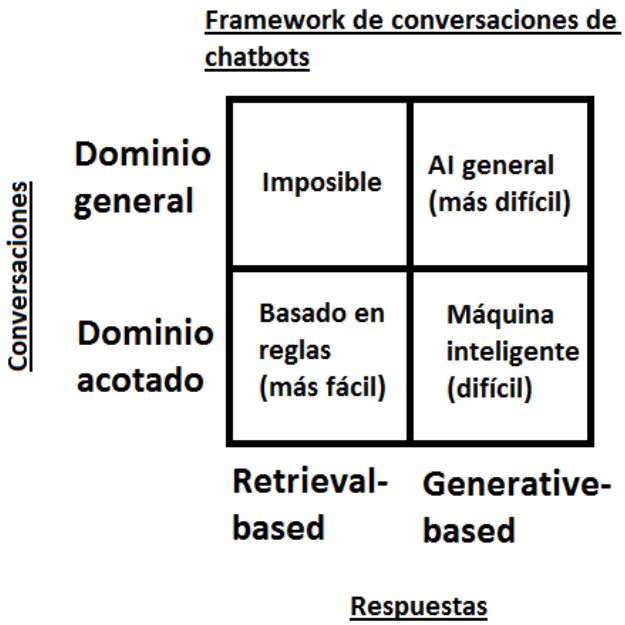


Figura 2.1: Classification of different chatbot models according to the domain and how to generate the answers

In this way the most important goals to build a good AI chatbot are:

- Integrate the context.
- Generate coherent answers.
- Choose and build the model well.
- Train it so that it has a good knowledge base.

To transform the language into actions and contexts we must use NLP (Natural language processing) with deep learning.

### 2.2.2. State of the art chatbots

There are several examples of chatbots already available on the network. We have researched and according to public opinion the best ones are among others: Mitsuku, Rose, Right Click, Poncho, Insomo bot, Dr A.I, Melody by baidu. The most important to analyze for our goal are Dr A.I and Melody by baidu since they are two medical chatbots.

Dr A.I : Among its features are having an empathetic conversation with the client about their symptoms and general health to know the context. Later, with the help of Artificial Intelligence and Machine Learning, Dr. A.I. immediately translates the symptoms into possible medical explanations that will

be customized according to age, gender, possible external conditions, medications, and other relevant aspects of the patient's health. Finally Dr A.I. gives an adequate medical treatment with the improvement of the patient, after all the process the user has the opportunity to speak with a real doctor.

Melody by baidu: This chatbot contrary to the previous one is thought for the doctor to use and not the patient. Provide the doctor with relevant information to assist the patient with useful recommendations and possible treatments. Melody incorporates Advanced Deep Learning and NLP. In addition doctors use Melody so that the patients themselves have it at home to get quick answers about their health problems.

## 2.3. Document's structure

After this introduction, chapter 3 describes the operation of the different techniques used (Machine Learning, Neural Networks and Deep Learning) both automatic learning and text analysis. In Chapter 4 we will explain the extraction of language properties and one of the most powerful free software NLP libraries: Spacy. In chapter 5 we will discuss the choice of computer language and execution environment. In the next chapter (6) we will explain our architecture for generating responses in the biomedical field, describing the process of adapting the different techniques to the specific problem and everything we have learned. Chapter 7 is devoted to the experiments carried out and the analysis of results. Finally, chapter 8 consists of the conclusions and an approach to future work.

# Capítulo 3

# Fundamentos técnicos

## 3.1. Machine Learning

El Machine Learning, o aprendizaje automático, es una rama de la inteligencia artificial que busca hacer que las máquinas sean capaces de aprender a partir de ejemplos. Existen dos tipos de **aprendizaje automático**:

- **Supervisado**: en el cual sabemos las características que se deberían cumplir en el output dado un input. Existen dos tipos de aprendizaje supervisado:
  - **Clasificación** (problema discreto): trata de solucionar un problema en el que la respuesta pertenece a un número finito (dos o más) de clases. Por ejemplo ¿mañana lloverá? Sí o no.
  - **Regresión** (problema continuo): mapea un conjunto de inputs continuos en otro conjunto de outputs continuos Por ejemplo ¿Cuántos $cm^3$ de agua caerán mañana?
- **No supervisado**: Se diferencia del aprendizaje supervisado en que no hay un conocimiento a priori de las características que deben tener los datos del output. Por ejemplo, coger una muestra de 1.000.000 de hormigas y que el algoritmo, solo, te las ordene por subespecies sin, la persona, tener conocimiento anterior de que subespecies de hormigas existen.

En todos los casos donde se use la técnica Machine Learning se deberá crear un **modelo de representación** de los datos que se usarán en el estudio. Para el caso de la lluvia basta con hacer una tabla con dos columnas con el día ($x^{(i)}$) y los $cm^3$ ($y^{(i)}$) de agua caídos:

| **X** | **Y** |
|---|---|
| 1 | 0 |
| 2 | 10 |
| 3 | 4 |
| 4 | 0 |
| 5 | 2 |

Esta tarea a menudo no es tan trivial.

### 3.1.1. Regresión lineal

La regresión lineal es una técnica estadística utilizada para calcular la función lineal que más se aproxima a un conjunto de datos. Calculamos esta función para poder predecir nuevas situaciones que no se encuentren en nuestro conjunto de datos inicial. Primero hablaremos de la regresión lineal para una variable, y más tarde generalizaremos para el caso de múltiples variables.

Para la regresión lineal con una variable, tras haber creado el modelo de representación, el siguiente paso consiste en inventar una **función de hipótesis $\mathbf{h(x^{(i)})}$** que consiga, con la mayor exactitud posible, dar el valor $y^{(i)}$ sin usar los valores del **conjunto de entrenamiento** (training set), que es nada más que la tabla anterior. Nuestra hipótesis consistirá por tanto de una función lineal (una recta), en la que obtendremos nuestra estimación de $y^{(i)}$ a partir de $x^{(i)}$. Esto es:

$$h_\theta(x^{(i)}) \;=\; \theta_0 + \theta_1 x^{(i)} \tag{3.1}$$

Lo que se puede representar más fácilmente mediante la multplicación de dos **vectores**.

$$h_\theta(x^{(i)}) = \begin{bmatrix}\theta_0 & \theta_1\end{bmatrix} \begin{bmatrix}1 \\ x^{(i)}\end{bmatrix} = \theta^T x \tag{3.2}$$

Y si extrapolamos al caso general para cada una de las $x^{(i)}$, obtenemos el vector $h_\theta(x^{(i)})$ a partir de la multiplicación de dos **matrices**, donde cada columna representa el ejemplo de entrenamiento $x^{(i)}$.

$$\begin{gathered} h_\theta(X) = \begin{bmatrix}\theta_0 & \theta_1\end{bmatrix} \begin{bmatrix}x^{(1)} x^{(2)} \dots x^{(m)}\end{bmatrix} = \theta^T X \\ \textit{donde } x^{(i)} \textit{ es el vector de la ecuación } 3{,}2 \end{gathered} \tag{3.3}$$

La matriz $\theta$ contendrá los **parámetros** que usaremos en $h(x^{(i)})$ para hallar el valor $y^{(i)}$, y la otra matriz **X** contendrá los vectores de 2 elementos de los ejemplos de entrenamiento $x^{(i)}$ en las filas. Por conveniencia el valor de $x_0$ será 1. Por ejemplo para $\mathbf{h_\theta(x) = -40 + 0{,}25x}$.

$$X = \begin{bmatrix}1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852\end{bmatrix} \theta = \begin{bmatrix}-40 \\ 0{,}25\end{bmatrix} \tag{3.4}$$

$$X\theta = \begin{bmatrix} 486 \\ 314 \\ 344 \\ 173 \end{bmatrix} \tag{3.5}$$

Para calcular la precisión de la **función de hipótesis** se usa una **función de coste.** La más usada es la **función de error cuadrática J($\theta_0$, $\theta_1$)** : que es la media de los cuadrados de las diferencias entre los resultados de la hipótesis y los del **conjunto de datos de test** (test set), concretamente es la mitad del **Error Cuadrático Medio (ECM)** estadístico. Esto se hace por conveniencia, ya que luego tendremos que hacer la derivada de la función.

$$J\left(\theta_0, \theta_1\right) = \frac{1}{2m} \sum \left(\hat{y}_i - y_i\right)^2 = \frac{1}{2m} \sum \left(h_\theta\left(x_i\right) - y_i\right)^2 \tag{3.6}$$

$$donde\ h_\theta\left(x_i\right) \ = \ \theta_0 \ + \theta_1 x_i$$

Una vez tengamos todo este trabajo realizado debemos seleccionar el conjunto de parámetros $\theta$ a usar en nuestra **función de hipótesis** y someterlos a prueba con nuestra función de coste para ver cuáles llegan al mínimo global. Para eso usamos el **método de optimización, descenso del gradiente,** que consiste en ir bajando los valores de la función hasta llegar a un mínimo local, de la siguiente manera:

Desde i = 1 hasta que converja en un mínimo local repetir:

$$\theta_j = \theta_j - \ \alpha \ \frac{\partial J(\theta_0, \ \theta_1)}{\partial \theta_j} \tag{3.7}$$

Donde, j=0,1 representa el índice del parámetro, $\alpha$ es un parámetro que debemos ajustar y la derivada parcial determina la dirección del siguiente paso.

Si damos a $\boldsymbol{\alpha}$ valores pequeños el descenso de gradiente se podría ralentizar demasiado, pero si nos pasamos puede no converger en el mínimo o incluso diverger y pasar a tener en cada iteración valores cada vez más grandes de $\mathbf{J(\theta_0, \theta_1)}$, que es lo contrario que nuestro objetivo.

De este modo, si hacemos la derivada parcial que nos falta por calcular la fórmula final sería:
Repetir hasta converger: {

$$\begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_0\left(x_i\right) - y_i\right) \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\left(h_0\left(x_i\right) - y_i\right) x_i\right) \end{aligned} \tag{3.8}$$

}

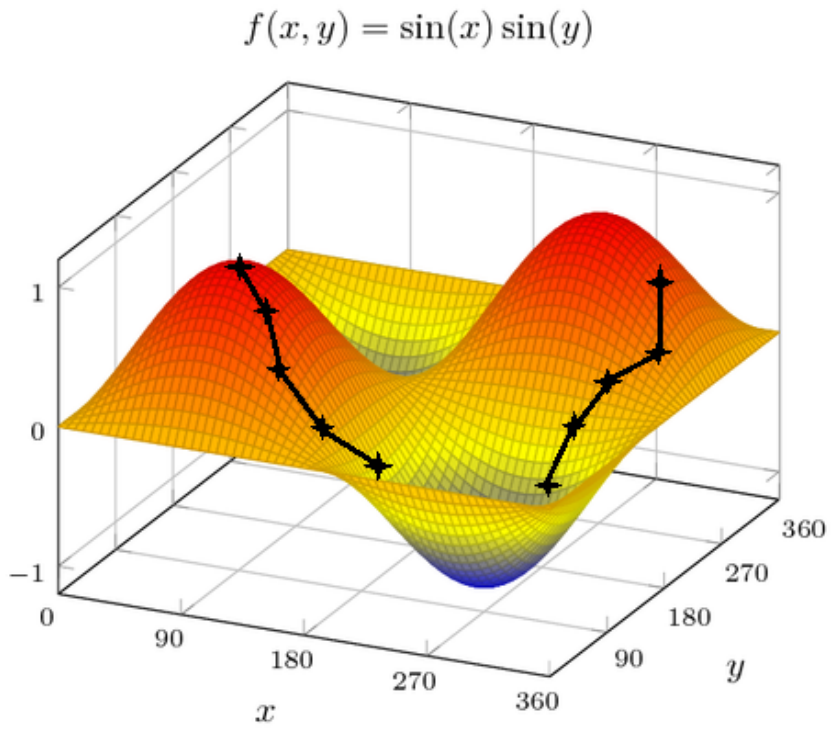


Figura 3.1: Representación gráfica del descenso de gradiente[1].

En el caso de tener más de dos variables, la función de hipótesis toma la siguiente forma:

$$h_\theta\left(x_i\right) \;=\; \theta^T x \;=\; \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n \tag{3.9}$$

[1]Esta imagen ha sido copiada y modificada. Nos hemos asegurado de que en google estaba etiquetada para reutilización con modificaciones. El autor de la gráfica es Stefan Kottwitz y nosotros hemos añadidos las cruces y líneas negras.

Donde $x$ es un vector de n + 1 elementos para el que $x_0$ siempre vale 1.

$$h_\theta(x) = \begin{bmatrix}\theta_0 \ \theta_1 \dots \ \theta_n\end{bmatrix} \begin{bmatrix}x_0 \\ x_1 \\ \vdots \\ x_n\end{bmatrix} = \theta^T x \tag{3.10}$$

De esta manera, la fórmula de descenso de gradiente sería:
Repetir hasta converger: {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)} \qquad \text{para j := 0...n} \tag{3.11}$$

}

Una manera de acelerar este cálculo es escalando las variables, concentrándolas en un rango más pequeño. Idealmente:

$$-1 \leq x_{(i)} \leq 1 \quad \text{ó} \quad -0{,}5 \leq x_{(i)} \leq 0{,}5 \tag{3.12}$$

Para ello restamos la media $\boldsymbol{\mu_i}$ de todos los valores y lo dividimos por el rango $\boldsymbol{s_i}$ (max - min).

$$x_i := \frac{x_i - \mu_i}{s_i} \tag{3.13}$$

Para comprobar que nuestro descenso de gradiente está bien implementado tenemos que asegurarnos de que los valores de $\mathbf{J(\Theta)}$ vayan bajando, sino tal como hacíamos en el descenso lineal tenemos que dar valores más pequeños a $\boldsymbol{\alpha}$. Lo ideal es tener dicho valor por debajo de un umbral $\boldsymbol{\varepsilon}$, generalmente $\mathbf{10^{-3}}$, aunque para la mayoría de proyectos será difícil poner ese límite.

Para mejorar el rendimiento de nuestra función podemos combinar varias características. Por ejemplo si combináramos $\boldsymbol{x_1}$ y $\boldsymbol{x_2}$ en $\boldsymbol{x_3}$, una posible manera sería:

$$x_3 = x_1 x_2 \tag{3.14}$$

De la misma manera si nuestra función lineal no funcionara de la manera que deseamos podríamos crear nuevas variables elevando las anteriores al cuadrado, al cubo, haciéndoles la raíz cuadrada, etc.

### 3.1.2. Problemas de clasificación

En este caso la salida que queremos predecir tan solo toma un pequeño número de valores discretos. Por el momento nos centramos en problemas de clasificación binaria; es decir, hay dos valores posibles (0 ó 1, − ó +, sí ó no . . . ). Dada una variable de ejemplo $x^{(i)}$, se conoce como etiqueta (label) al correspondiente

valor de $y^{(i)}$.

Podemos abordar el problema de clasificación ignorando el hecho de que $y$ sólo toma valores discretos, y usar el algoritmo de regresión lineal con un pequeño ajuste, ya que no tiene sentido que $h_\theta(x)$ tome valores por encima de 1 ni por debajo de 0. Para satisfacer $0 \leq h_\theta(x) \leq 1$, aplicamos una **función sigmoide** a nuestra hipótesis anterior, concretamente la **función logística**:

$$g(z) = \frac{1}{1 + e^{-z}} \tag{3.15}$$

Esta función satura en las colas, concretamente tiene una asíntota horizontal en $y = 1$ al tender a $+\infty$, y otra en $y = 0$ al tender a $-\infty$.

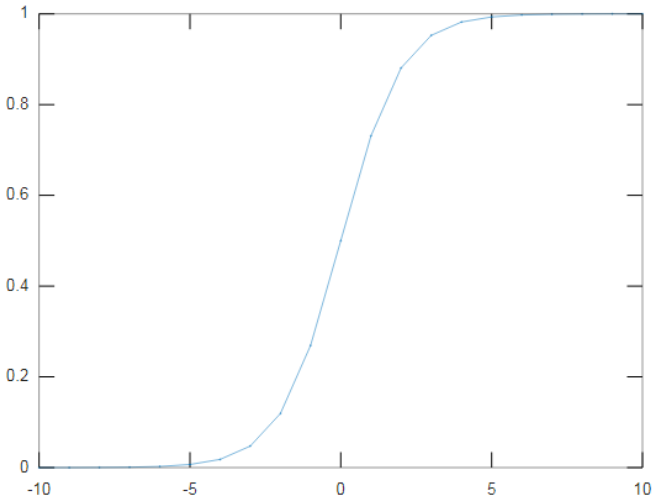


Figura 3.2: Función sigmoide.

Así, si nuestra función de hipótesis era $h_\theta(x) = \theta^T x$, definimos una nueva función de hipótesis $h'_\theta(x)$ mediante la composición de nuestra hipótesis anterior $h_\theta$ con la función logística $g$:

$$h'_\theta(x) = (h_\theta \circ g)(x) = g(h_\theta(x)) = \frac{1}{1 + e^{-\theta^T x}} \tag{3.16}$$

A partir de ahora llamaremos directamente $h_\theta$ a esta nueva función de hipótesis $h'_\theta$. Podemos decir pues que $h_\theta$ nos da la **probabilidad** de que nuestra salida sea 1.

Para obtener una clasificación discreta de la salida (0 ó 1), hacemos la siguiente distinción:

$$\begin{aligned} h_\theta(x) \geq 0{,}5 &\longrightarrow y = 1 \\ h_\theta(x) < 0{,}5 &\longrightarrow y = 0 \end{aligned} \tag{3.17}$$

Sabemos que:

$$\begin{aligned} h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} \geq \frac{1}{2} \iff 2 \geq 1 + e^{-\theta^T x} \iff \\ \iff 1 \geq e^{-\theta^T x} \iff \log(1) \geq -\theta^T x \iff \theta^T x \geq 0 \end{aligned} \tag{3.18}$$

De donde deducimos:

$$\begin{aligned} \theta^T x \geq 0 \longrightarrow y = 1 \\ \theta^T x < 0 \longrightarrow y = 0 \end{aligned} \tag{3.19}$$

Llamamos **frontera de decisión** (decission boundary) a la línea que separa el área donde $y = 0$ e $y = 1$.

Podemos extender esta forma de resolver problemas de clasificación a la situación en que tengamos más de dos categorías. En vez de $y = 0, 1$ expandimos a $y = 0, 1 \dots n$. Para ello simplemente planteamos $n$ subproblemas, donde cada uno es el problema de clasificación binaria "uno contra el resto", es decir, medimos la probabilidad de que $y$ sea miembro de una de nuestras clases frente a que pertenezca a cualquiera de las demás.

$$\begin{aligned} & y \in \{0, 1...n\} \\ & h_\theta^{(0)}(x) = P(y = 0|x; \theta) \\ & h_\theta^{(1)}(x) = P(y = 1|x; \theta) \\ & \dots \\ & h_\theta^{(n)}(x) = P(y = n|x; \theta) \\ & \text{predicción} = \max_i(h_\theta^{(i)}(x)) \end{aligned} \tag{3.20}$$

### 3.1.3. Modelo de regresión logística

Al contrario que con la regresión lineal, donde nuestra función de hipótesis era lineal ($h_\theta(x) = \theta^T x$), ahora nuestra función de hipótesis es la función logística. Al no ser lineal, nos va a ser más difícil buscar el mínimo si le aplicamos nuestra función de coste (ECM), ya que probablemente resultará en una función ondulante con muchos máximos y mínimos locales. Para solucionar esto definimos una nueva función de coste:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{Coste}(h_\theta(x^{(i)}), y^{(i)}) \tag{3.21}$$

$$Coste(h_\theta(x^{(i)}), y^{(i)}) = \begin{cases} -\log(h_\theta(x^{(i)}), & \text{si } y = 1. \\ -\log(1 - h_\theta(x^{(i)}), & \text{si } y = 0. \end{cases} \tag{3.22}$$

Se puede observar que con esta nueva función de coste, si $y = 0$ (respectivamente $y = 1$) y $h_\theta(x) \longrightarrow 1$ (resp. $h_\theta(x) \longrightarrow 0$), la función Coste se va al infinito. Esto es porque queremos penalizar altamente las predicciones erróneas.
Podemos especificar nuestra función de coste 3.22 como una única ecuación discriminando según el valor de $y^{(i)}$:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))] \tag{3.23}$$

O, en notación vectorial:

$$\begin{aligned} &h = g(X\theta) \\ &J(\theta) = \frac{1}{m}\cdot\left(-y^T\log(h) - (1-y)^T\log(1-h)\right) \end{aligned} \tag{3.24}$$

Si a esto le aplicamos la fórmula general de descenso de gradiente y resolvemos, obtenemos un algoritmo idéntico al utilizado para regresión lineal:

$$\begin{aligned} &\textit{Repetir } \{ \\ &\theta_j := \theta_j - \frac{\alpha}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} \\ &\} \end{aligned} \tag{3.25}$$

Que en notación vectorial se puede expresar como:

$$\theta := \theta - \frac{\alpha}{m}X^T(g(X\theta) - \vec{y}) \tag{3.26}$$

## 3.2. Redes Neuronales

Las redes neuronales están inspiradas en el comportamiento biológico de las neuronas y en como se organizan formando la estructura del cerebro. Esquemáticamente, una neurona se compone de **dendritas**, por donde recibe la información, un **núcleo** donde procesa la información, y un **axón** por donde envía la información procesada a las siguientes neuronas (que reciben por sus dendritas, procesan en su núcleo y vuelven a enviar...).

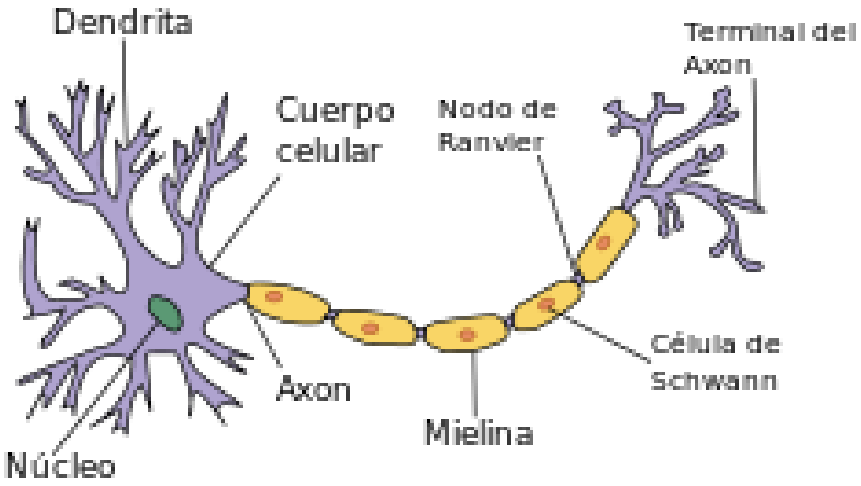


Figura 3.3: Estructura básica de una neurona del sistema nervioso[2].

En nuestro modelo, las dendritas serán nuestras variables de entrada $x_1, x_2, \ldots, x_n$, y la salida del axón procesada por el núcleo será la salida de nuestra función de hipótesis (también llamada **función de activación**). En este contexto, nuestro vector de parámetros $\theta$ es también llamado **vector de pesos**. Las redes neuronales se componen de una primera capa (*input layer*), donde están los nodos de entrada, una o más capas intermedias llamadas capas ocultas (*hidden layers*) y una capa de salida (*output layer*) de donde obtenemos la predicción final. Los nodos de las capas ocultas, también llamados nodos de activación los denotaremos $a_i^{(j)}$, que es la unidad de activación $i$ en la capa $j$. Así, una red neuronal sencilla con una única capa oculta de tres nodos de activación y tres variables de entrada podría ser de la forma:

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix} \rightarrow h_\theta(x)$$

También llamaremos $\Theta^{(j)}$ a la matriz de paso del nivel $j$ al $j+1$. Esta matriz tendrá tantas filas como nodos tenga la siguiente capa, y tantas columnas como nodos tenga la capa actual más uno (el nodo de sesgo, $\Theta_0^{(j)}$, que siempre será multiplicado por $x_0 = 1$). Con todo esto definimos el valor de cada nodo de activación como:

$$\begin{aligned} a_1^{(2)} &= g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3) \\ a_2^{(2)} &= g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3) \\ a_3^{(2)} &= g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3) \\ h_\Theta(x) = a_1^{(3)} &= g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}) \end{aligned} \tag{3.27}$$

[2]Neurona, por Wikipedia, licenciado bajo CC BY-SA 3.0

De una forma más general, si definimos:

$$z_k^{(j)} = \Theta_{k,0}^{(j-1)} x_0 + \Theta_{k,1}^{(j-1)} x_1 + \cdots + \Theta_{k,n}^{(j-1)} x_n \tag{3.28}$$

Podemos expresar nuestras unidades de activación como:

$$a_k^{(j)} = g(z_k^{(j)}) \tag{3.29}$$

Vectorialmente tenemos:

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad z^{(j)} = \begin{bmatrix} z_1^{(j)} \\ z_2^{(j)} \\ \vdots \\ z_n^{(j)} \end{bmatrix} \tag{3.30}$$

Y si llamamos $a^{(1)}$ al vector de entrada $x$, podemos reescribir la ecuación como:

$$z^{(j)} = \Theta^{(j-1)} a^{(j-1)} \tag{3.31}$$

Así, poniéndolo todo junto obtenemos que el valor de nuestra función de hipótesis es:

$$h_\Theta(x) = a^{(j+1)} = g(z^{(j+1)}) = g(\Theta^{(j)} a^{(j)}) \tag{3.32}$$

### 3.2.1. Función de coste de las redes neuronales

Antes que nada definiremos las nuevas variables que vamos a usar:

- L = Número total de capas en la red.
- $S_l$ = Número de unidades (sin contar la unidad bias) en la capa l.
- K = Número de clases en las que clasificamos.

Diferenciaremos entre dos tipos de problema:

Clasificación binaria: ($S_l = 1$, K = 2)

$$y = 0 \quad ó \quad 1 \qquad\qquad h_\Theta(X) \in R \tag{3.33}$$

Clasificación multiclase (K clases):

$$y \in R^K \qquad\qquad h_\Theta(X) \in R^K \tag{3.34}$$

$$J(\Theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[ y_k^{(i)} \log((h_\Theta(x^{(i)}))_k) + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{j,i}^{(l)})^2 \tag{3.35}$$

Se han añadido algunos sumatorios anidados para dar cuenta de los múltiples nodos de output. El doble sumatorio suma los costes de la regresión logística calculados por cada neurona en la capa de output y el triple sumatorio suma los cuadrados de cada parámetro individual en la red neuronal entera.

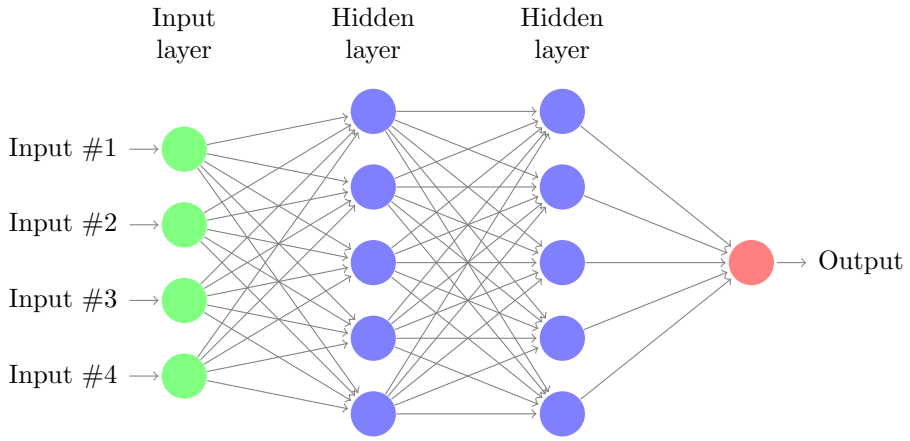


Figura 3.4: Grafo de una red neuronal standard.

### 3.2.2. Algoritmo de retropropagación

#### 3.2.2.1. Descripción del algoritmo

Con las redes neuronales calcular la derivada parcial de los errores de cada neurona es muy costoso y para esto se usa el algoritmo de retropropagación.

Antes de seguir definiremos:

- $\delta_j^{(L)}$ = "error" del nodo j en la capa l.

Si nos fijamos en la imagen de arriba para cada unidad de output (capa L = 4):

$$\delta_j^{(4)} = a_j^{(4)} - y_j \quad dónde \quad a_j^{(4)} = (h\Theta(x))_j \tag{3.36}$$

Para los errores anteriores podemos usar el $\delta_j^{(l)}$, del último, de la siguiente manera:

$$\delta_j^{(3)} = (\Theta^{(2)})^T \delta_j^{(4)} .* g'(z^{(3)}) \quad dónde \quad g'(z^{(3)}) = a^{(3)} .* (1 - a^{(3)}) \tag{3.37}$$

$$\delta_j^{(2)} = (\Theta^{(3)})^T \delta_j^{(4)} .* g'(z^{(2)}) \quad dónde \quad g'(z^{(2)}) = a^{(2)} .* (1 - a^{(2)}) \tag{3.38}$$

Una vez tengamos calculado $\delta_j^{(L)}$, calcularemos la acumulación del "error", con la siguiente fórmula:

$\Delta_{i,j}^{(l)} := 0$ (para todo l, i, j), para darle después los siguientes valores:

$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$ que se vectoriza a $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$

Después añadimos nuestros parámetros de regularización en otra matriz D, teniendo en cuenta que cuando j=0 estamos ante la neurona que siempre vale 1 (bias).

$$D_{i,j}^{(l)} := \frac{1}{m}\left(\Delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(l)}\right) \qquad \text{para j} > 0 \tag{3.39}$$

$$D_{i,j}^{(l)} := \frac{1}{m}\Delta_{i,j}^{(l)} \qquad \text{para j} = 0 \tag{3.40}$$

Una vez hayamos hecho bien los cálculos obtendremos:

$$\frac{\partial}{\partial\Theta_{i,j}^{(l)}}J(\Theta) \tag{3.41}$$

Ahora veremos por qué eso es igual a la derivada parcial del coste. Recordemos la función de coste para redes neuronales:

$$J(\Theta) = -\frac{1}{m}\sum_{t=1}^{m}\sum_{k=1}^{K}\left[y_k^{(t)}\ \log(h_\Theta(x^{(t)}))_k + (1-y_k^{(t)})\ \log(1-h_\Theta(x^{(t)})_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(\Theta_{j,i}^{(l)})^2 \tag{3.42}$$

Si consideramos una clasificación de una sola clase (k = 1) y no hacemos regularización, el coste se queda como:

$$cost(t) = y^{(t)}\ \log(h_\Theta(x^{(t)})) + (1-y^{(t)})\ \log(1-h_\Theta(x^{(t)})) \tag{3.43}$$

Intuitivamente $\delta_j^{(l)}$ es el "error" para $a_j^{(l)}$ (unidad j en la capa l). Más formalmente, los valores de delta son en realidad las derivadas de la función de coste:

$$\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}}coste(t) \tag{3.44}$$

## 3.3. Sequence to sequence

Sequence to sequence es un tipo de problema de aprendizaje que trata de entrenar modelos para convertir secuencias de un dominio (por ejemplo, frases en inglés), a secuencias de otro dominio (por ejemplo, la misma frase en español). Esto puede ser usado tanto para traducción, como pone en nuestro ejemplo, como para QA (question answering), que es el problema que nos ocupa. Además,

es aplicable a cualquier situación en la que debemos generar texto. Hay diversas maneras de afrontar este problema pero nosotros nos centraremos en RNNs (Recurrent Neural Networks), que serán explicadas a lo largo de este capítulo, dado a que son las que en la actualidad están funcionando mejor.

## 3.4. ¿Por qué no usaremos las redes neuronales standard?

Hay dos explicaciones principales por las que se ha tenido que utilizar otro modelo de redes neuronales para solucionar este problema:

- El input y el output pueden tener diferentes tamaños en diferentes ejemplos. Por ejemplo, la respuesta de una pregunta del training set puede ser más larga que otra pregunta del mismo training set, esto es muy común.
- Las redes neuronales standard no comparten propiedades aprendidas entre diferentes posiciones del texto. Sin embargo, para nosotros, el orden de las palabras es importante y debemos de poder relacionar esas posiciones con el significado final de una frase.

## 3.5. Recurrent Neural Networks: RNNs

Las redes neuronales recurrentes permiten representar inputs secuenciales de un tamaño arbitrario en forma de vectores de una longitud fija, también tiene la capacidad de extraer propiedades estructurales de los inputs. Por ejemplo, ¿en qué contexto es más probable que aparezca la palabra "wonderful"? RNNs, particularmente las compuestas por gated architectures, como las LSTM, son muy potentes extrayendo regularidades estadísticas en inputs secuenciales.

Usaremos $x_{i:j}$ para referirnos a la secuencia de vectores $x_i, \ldots, x_j$. En general, una red neuronal recurrente es una función que toma como input una secuencia ordenada de longitud arbitraria de n vectores de dimensión $d_{in} \quad x_{1:n} = x_1, x_2, \ldots, x_n, (x_i \in \mathbb{R})$ y devuelve como output un único vector de dimensión $d_{out} \quad (y_n \in \mathbb{R}^{d_{out}})$:

$$y_n = RNN(x_{1:n}) \tag{3.45}$$

Este sistema implícitamente define un vector de output $y_i$ por cada subsecuencia $x_{1:i}$ de la secuencia $x_{1:n}$. Llamaremos RNN* a la función que retorna esta secuencia:

$$y_{1:n} = RNN^*(x_{1:n}) \quad dónde \quad y_i = RNN(x_{1:i}) \tag{3.46}$$

Si entramos en más detalle, la red neuronal recurrente está definida recursivamente, una función R que tiene como input el vector de estado $s_{i-1}$ y un vector de input $x_i$ que retorna un nuevo estado $s_i$. Este vector de estado $s_i$ es mapeado a un vector de output $y_i$ usando una función O. La base de la recursión es un

vector de estado inicial $s_0$, que es también un input de la RNN. Este vector suele contener solamente ceros. Al construir una RNN se debe especificar la dimensión de los inputs $x_i$ y de los outputs $y_i$.

$$RNN^*(x_{1:n}; s_0) = y_{1:n}$$
$$y_i = RNN(x_{1:i})$$
$$s_i = R(s_{i-1}, x_i)$$

Las funciones R, O son las mismas a lo largo de la secuencia, pero la RNN mantiene un seguimiento de los estados de la computación mediante el vector de estado $s_i$ que se mantiene y es insertado en cada invocación de R.

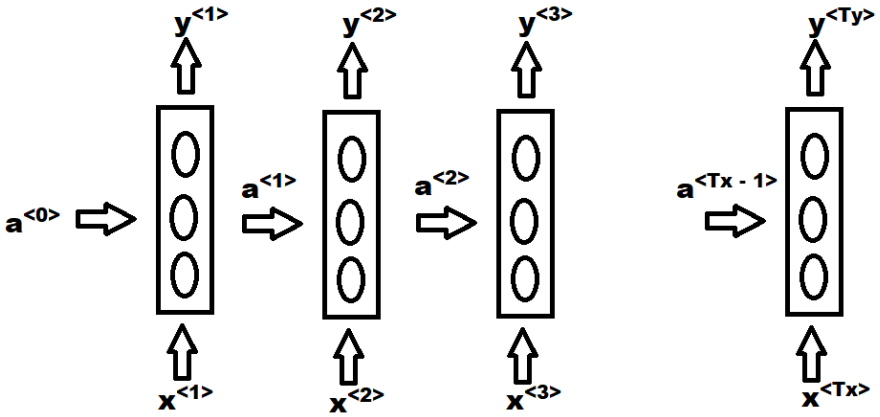


Figura 3.5: RNN

## 3.6. Modelos comunes de RNNs

Este capítulo es muy importante para poder comprender cómo hemos hecho el sistema de búsqueda de respuestas final.

### 3.6.1. Acceptor

Algunas veces, al usuario de las RNNs, solamente le interesa el valor del último vector de output, $y_n$. Viéndolo así, la RNN se estaría entrenando como un aceptor (clasificador). Observamos el estado final y decidimos cuál es el output. Por ejemplo, imaginemos entrenar una RNN que lee una frase y, basado en el estado final decide si expresa un sentimiento negativo o positivo. La función de coste en estos casos se define en términos de la función $y_n = O(s_n)$. Normalmente, el vector de output $y_n$ de la RNN se le pasa como parámetro a una red neuronal standard que produce una predicción.

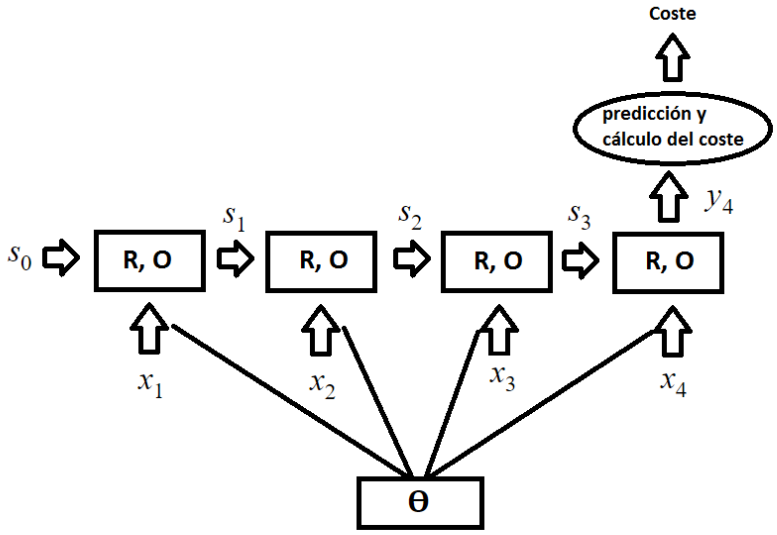


Figura 3.6: Grafo del entrenamiento de una red RNN Acceptor

### 3.6.2. Encoder

Al igual que el acceptor, un encoder usa solamente el vector de output, $y_n$. Aún así, al contrario que el acceptor, donde la predicción se hace solamente a partir del vector de output, aquí este vector no pasa por una red neuronal standard, y se usa para codificar la secuencia de entrada, que sirve como información adicional junto a otras señales de input. Por ejemplo, para un sistema de resumenes de textos, primero se debe ejecutar una RNN sobre el documento, para obtener así, un vector $y_n$ que contiene la información más relevante de todo el texto. Después este último vector, será utilizado junto con otros parámetros para seleccionar las frases que se deben incluir en el resumen.

### 3.6.3. Transducer

Otra opción es producir un output $\hat{t}_i$ para cada input que lee. De esta manera, podemos computar un coste local $L_{local}(\hat{t}_i, t_i)$, para cada predicción $\hat{t}_i$ basado en el output real $t_i$. El coste para la secuencia entera será: $L(\hat{t}_{1:n}, t_{1:n}) = \sum_{i=1}^{n} L_{local}(\hat{t}_i, t_i)$, o usando cualquier otro tipo de combinación como, por ejemplo, la media. Un caso especial de RNN Transducer es la RNN Generator, este patrón se explicara en parte en la sección 3.9 y la otra parte en la sección 6.3.

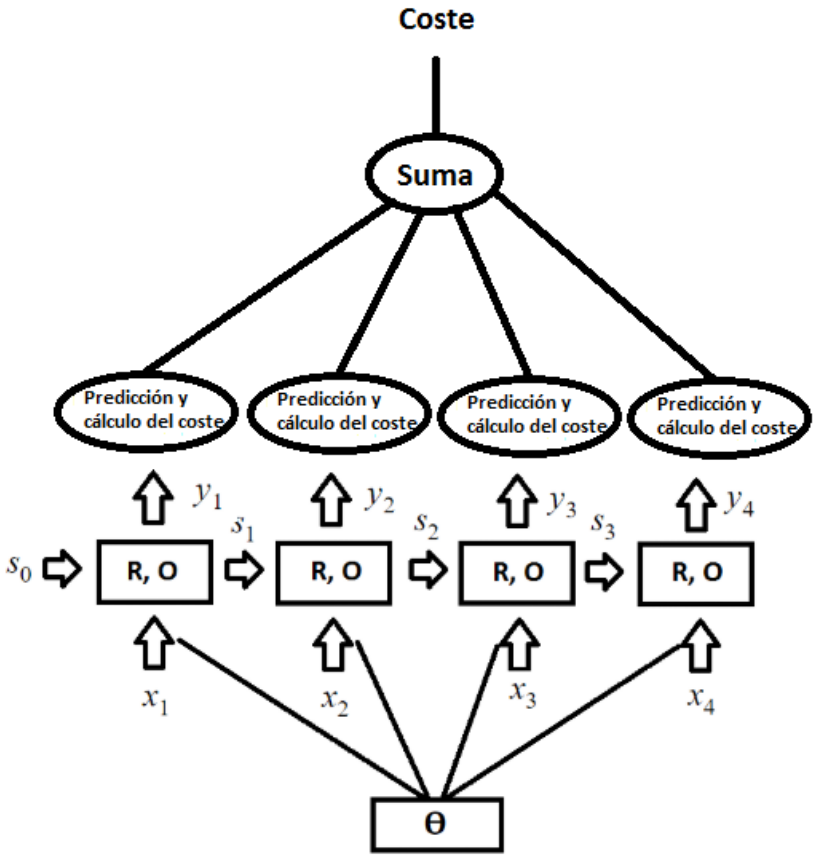


Figura 3.7: Grafo del entrenamiento de una red RNN Transducer

## 3.7. Bidirectional RNNS (BIRNN)

Una modificación muy útil de las RNNs son las redes neuronales recurrentes bidireccionales. Consideremos una secuencia como input $x_{1:n}$. La biRNN funciona manteniendo dos estados separados, $s_i^f$ y $s_i^b$ para cada posición del input i. El estado de avance, $s_i^f$, esta basado en la subsecuencia $x_1, x_2, \ldots, x_i$ mientras que el estado de retroceso, $s_i^b$, está basado en la subsecuencia $x_n, x_{n-1}, \ldots, x_i$. Los dos estados son calculados por dos RNNs diferentes. El estado $s_i$ es la concatenación del estado de avance y del estado de retroceso.

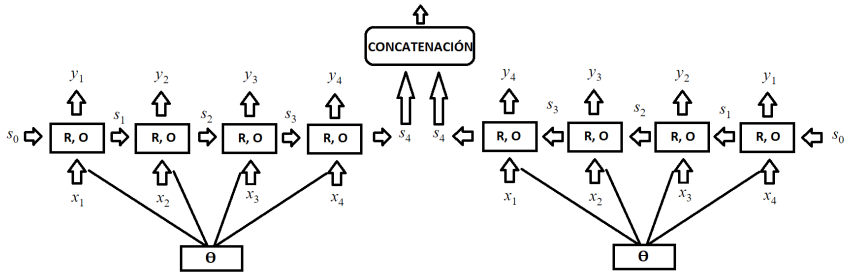


Figura 3.8: Bidirectional RNN

## 3.8. Gated architectures

Debido al funcionamiento intrínseco de las redes neuronales recurrentes que se componen de un nodo por cada elemento de la secuencia de input, o incluso más, esta longitud ocasiona que, como se actualizan los pesos para dar poder a las propiedades más importantes y menos poder a las menos importantes, para la solución de nuestro problema, en cada uno de los nodos, puede que se exceda. Este exceso hace que hayan propiedades que directamente desaparezcan (vanishing gradients problem) y otras tomen una importancia excesiva que no concuerda con la realidad (exploding gradients problem). Esto ocurre porque el input se multiplica con los pesos en cada nodo por lo que al final tendríamos:

$$y = w_1 * w_2 * \cdots * w_n * x \tag{3.47}$$

Al prestar atención a la fórmula anterior vemos que a los primeros elementos de la secuencia afectan más multiplicaciones de pesos y esto provoca una pérdida de información, y por lo tanto, memoria para estos elementos. Para solucionar este problema hace falta modificar las RNNs simples para controlar, en cada nodo, cuanto input interesa insertar y cuanto interesa mantener del vector del estado que representa la memoria.

### 3.8.1. Long Short Term Memory (LSTM)

La arquitectura LSTM fue diseñada para solucionar el vanishing gradient problem, y fue la primera en introducir el mecanismo de puertas. La red LSTM parte el vector de estado $s_i$ por la mitad. Una mitad es tratada como "celdas de memoria" y la otra como memoria de trabajo. Las celdas de memoria están diseñadas para preservar la memoria, además de ejecutar el descenso de gradiente, y están controladas por distintas puertas, que usan funciones matemáticas para simular puertas lógicas. En cada nodo se usa una puerta para decidir cuánto del nuevo input debe ser escrito en la celda de memoria y cuanto del actual contenido de la celda debe ser olvidado. Para comprender mejor el funcionamiento de la arquitectura es necesario observar la definición matemática de la misma:

$$
\begin{aligned}
s_j = R_{LSTM}(s_{j-1}, x_j) = [c_j; h_j] \\
c_j = f \odot c_{j-1} + i \odot z \\
h_j = o \odot tanh(c_j) \\
i = \sigma(x_j W^{xi} + h_{j-1} W^{hi}) \\
f = \sigma(x_j W^{xf} + h_{j-1} W^{hf}) \\
o = \sigma(x_j W^{xo} + h_{j-1} W^{ho}) \\
z = tanh(x_j W^{xz} + h_{j-1} W^{hz}) \\
y_j = O_{LSTM}(s_j) = h_j
\end{aligned}
$$

## 3.9. Encoder-Decoder

El encoder-decoder es una arquitectura que usa RNNs como un generador condicional. Esta arquitectura se divide en dos:

- Encoder: se encarga de codificar el input de la red neuronal para representar lo que se le conoce como contexto en un vector c. Este vector se puede calcular de diferentes maneras dependiendo del problema, lo más importante es que contenga la información más relevante del input. Ya sea una imágen, música o texto.
- Decoder: Sirve para predecir qué palabra irá después de una secuencia de palabras dada. Es un sistema que, dado un vector de contexto $c$ y una secuencia de palabras, predice la siguiente palabra de la frase, respuesta, resumen, etc.

La definición formal sería la siguiente:

$$
\begin{aligned}
p(w_{i+1} = k | w_{1:i}) = sigmoid(RNN(v_{1:i})) \\
v_j = [w_j; c]
\end{aligned}
$$

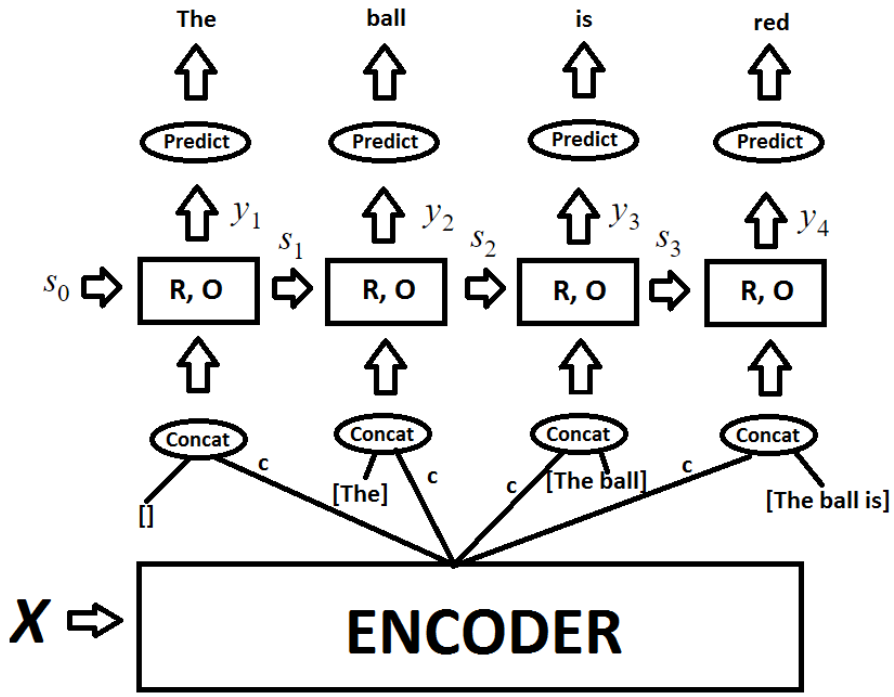


Figura 3.9: Grafo de la arquitectura Encoder-Decoder

# Capítulo 4

# Extracción de propiedades del lenguaje natural

En el capítulo 3 hemos explicado el problema de aprendizaje general, y hemos visto algunos modelos de Machine Learning y como entrenarlos. Todos estos modelos toman como input vectores $x$ y producen predicciones. Hasta ahora hemos asumido que esos vectores ya existían. En el procesamiento del lenguaje, los vectores $x$ son calculados a partir del texto que hay en nuestros datos, esto se hace para reflejar las propiedades lingüísticas del mismo. El mapeo del texto a vectores con valores en $\mathbb{R}$, se le denomina, extracción de propiedades. Estas secuencias necesitan ser convertidas en vectores y la forma en la que se hace no es trivial.

En este capítulo nos separaremos de toda la maquinaria del entrenamiento para explicar las propiedades del lenguaje más importantes y como representarlas en vectores. Nos centratemos en Spacy que es una de las mejores librerías que existen para adquirir tanto estas propiedades, como directamente los vectores de las palabras.

## 4.1. La importancia de la extracción de las propiedades en Natural Language Processing

- Palabra: En el procesamiento del lenguaje nos encontraremos palabras como: “dog”, “box”, o “hablar” y necesitaremos con nuestra representación vectorial decir algo sobre ellas. ¿Es un ser vivo? ¿En qué idioma está escrita? ¿Qué otras palabras tienen un significado similar a ella? ¿Es una palabra mal escrita que se refiere a otra? y más cosas. Este tipo de problemas son muy complicados. Por ejemplo, las palabras rara vez aparecen solas, por lo que se tiene que saber cuantas otras palabras del contexto se necesitan para saber su significado o poder representarlas en un vector.
- Textos: Dado un texto, ya sea un párrafo, una frase o un documento, podríamos necesitar decir algo sobre ellos. ¿Es spam o no? ¿Trata de viajes o de ciencia? ¿Es sarcástico? y así muchas más cosas. Este tipo de

preguntas son muy frecuentes y se les conoce como problemas de clasificación de documentos.

- Parejas de textos: Si nos dan una pareja de palabras o textos más largos, necesitamos decir algo sobre la pareja. ¿Son A y B sinónimos? ¿Es A una traducción válida de B? ¿Los textos A y B están escritos por el mismo autor?

- Palabra en un contexto: Si se parte de un trozo de texto y una palabra en particular de ese trozo, deberíamos poder representar esa palabra en función de su contexto. Por ejemplo, ¿la palabra "change" en la siguientre frase, "I want to change a colour", es un verbo, un sustantivo o un adjetivo? ¿Y en la frase "Here you have the change"?. Otro ejemplo sería, ¿"apple" se refiere a una fruta o una compañía en esta frase?, etc.

- Relación entre dos palabras: Aquí recibimos dos palabras o frases en el contexto de un documento más largo y necesitaríamos decir algo más sobre ellas. ¿Es A el sujeto de B? ¿Son A y B, el vendedor y el cliente en una compra?

## 4.2. Extracción de propiedades del lenguaje natural con Spacy

Spacy es una librería de código abierto para el procesamiento avanzado de lenguaje natural en Python. Está específicamente diseñada para uso en producción y ayuda a construir aplicaciones que procesan grandes cantidades de texto. Puede ser usado para extraer información de documentos, sistemas de entendimiento del lenguaje natural y para pre-procesar texto para el deep learning. Explicaremos todos los servicios que ofrece la librería pero nosotros lo usaremos específicamente para el pre-procesamiento del texto, más concretamente usamos el tokenizador y la función de similitud que ofrece el modelo "en_core_web_md". Este modelo es un paquete que ofrece Spacy. Durante el proceso de construcción también utilizamos los vectores que ofrece este paquete pero, debido a la no se ajustaban a nuestro modelo, finalmente descartamos su uso.

Entre los servicios que ofrece la librería, algunos hacen referencias a conceptos lingüísticos, mientras otros están relacionados con funcionalidades generales del machine learning:

- Tokenización: Segmenta el texto en palabras, signos de puntuación, etc. A estos segmentos se denominan tokens.

- Etiquetado Part-of-speech (POS): Asigna tipos de palabras a los tokens, como por ejemplo, nombre o verbo.

- Análisis de dependencias (Dependency Parsing): Asigna etiquetas de dependencia sintáctica, describiendo las relaciones entre tokens individuales, tales como sujeto o predicado.

- Lematización: Asignar las formas básicas de las palabras. Por ejemplo, el lemma de "was" es "be" y el de "rats" es "rat".

- Detección de los límites de las frases: Encuentra y segmenta todas las frases.
- Reconocimiento de entidades nominales: Etiqueta los nombres propios en personas, compañías o localidades.
- Similitud: Compara palabras, fragmentos de texto y documentos, y sabe calcular cuán similares son entre ellos.
- Clasificación de textos: asigna categorías y etiquetas a documentos enteros, o partes del mismo.
- Correspondencia basada en reglas: Funcionalidad similar a las expresiones regulares.
- Training: Actualiza y mejora predicciones de un modelo estadístico.
- Serialización: Guarda objetos en archivos o strings de bytes.

### 4.2.1. Part of speech (POS)

En la gramática tradicional inglesa, un POS, es una categoría de palabras o elementos léxicos que tienen propiedades gramaticales parecidas. Las palabras pertenecen al mismo part of speech si tienen un mismo comportamiento en términos de sintaxis. La lista de los part of speech que existen en inglés son: sustantivo, verbo, adjetivo, adverbio, pronombre, etc.

Después del proceso de tokenización, Spacy puede analizar y etiquetar un objeto dado que contiene información sobre el texto. El modelo estadístico de la librería le permite hacer una predicción de cuál es la etiqueta que mejor encaja para cada palabra en el documento pasado como parámetro.

### 4.2.2. Analizador de dependencias

Spacy tiene un analizador preciso y rápido de dependencias sintácticas, y una API para usar el árbol que se deriva del análisis. El analizador también detecta los límites de las frases, y permite iterar sobre noun phrases o "chunks".

Los noun chunks son sintagmas nominales básicas. Se puede entender un noun chunk como un nombre más una palabra que describe a ese nombre. Por ejemplo, "Autonomous cars".

Spacy da la oportunidad de deshabilitar el analizador de dependencias en caso de que no necesitemos información sintáctica alguna. Esto hará que la librería cargue y se ejecute mucho más rápido. También es posible habilitar el analizador y deshabilitarlo para algunos documentos en concreto.

### 4.2.3. Reconocimiento de entidades nominales

El modelo por defecto de Spacy identifica un variado conjunto de nombres propios o entidades numéricas, incluyendo compañías, localidades, organizaciones

y productos. Permite añadir clases al sistema de reconocimiento, y actualizar el modelo con nuevos ejemplos.

Una entidad nominal es un objeto del mundo al que se le asigna un nombre propio. Por ejemplo, una persona, un país o un libro. Spacy reconoce varios tipos de entidades nominales en un documento, llamando al modelo para recivir una predicción. Hay muchos tipos, entre ellos destacamos: PERSON, ORG, EVENT, etc. Puesto que los modelos son estadísticos y fuertemente dependientes de los ejemplos con los que ha sido entrenado, esto no siempre funciona a la perfección y por lo tanto puede ser necesario algún tipo de modificación por parte del usuario, dependiendo de su uso.

## 4.3. Word embeddings

Cuando se trabaja con lenguaje natural vemos que el mismo tiene propiedades, tales como palabras, letras o número. En esta sección explicaremos el modo de codificar estas propiedades de tal manera que podamos usarlas para clasificadores estadísticos. Plantearemos dos opciones, one-hot encoding y dense embedding vectors.

### 4.3.1. One-hot encodings

Esta codificación consiste en asignar un vector binario de dimensión igual al número de palabras que hay en el vocabulario. Lo haremos de la siguiente manera:

1. Daremos un id único a cada palabra del vocabulario, siendo el id más bajo 0 y el más alto n (siendo n el número de palabras).
2. Crearemos un vector de ceros de tamaño n.
3. Dada una palabra asignaremos, a la posición igual al id de la palabra, un 1.
4. Repetiremos el proceso para cada palabra.

El inconveniente de esta codificación es que no hay una relación clara entre el significado y uso de cada palabra.

### 4.3.2. Dense embedding vectors

Este tipo de codificación pretende representar cada una de las propiedades de cada palabra en su propio vector de dimensión fija d. Normalmente el vector d es mucho menor al número de propiedades de la palabra y suele tener entre 100 y 300 elementos. Estos vectores (embeddings) se pasan como parámetros a la red neuronal en forma de matriz.

Estos vectores se consiguen de la siguiente manera:

1. Extraemos un conjunto de propiedades clave $f_1, \ldots, f_n$ que sean relevantes para nuestro problema.

2. Para cada propiedad $f_i$, obtendremos un vector $v(f_i)$.
3. Combinaremos los vectores (ya sea por concatenación, suma o una combinación de ambos) en un vector de input x.
4. Añadiremos el vector x en una matriz de embeddings E.

Al mostrar estos vectores en una gráfica 3D se debería ver reflejado que las palabras relacionadas entre sí estén cerca una de la otra.

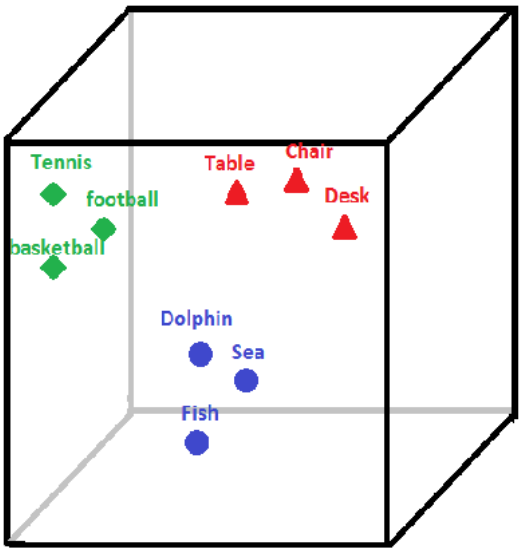


Figura 4.1: Gráfica en 3D que representa la relación de las palabras según dense embedding vectors.

## 4.4. Neural language models

Los modelos de lenguaje tradicionales sufrían una falta de generalización frente a diferentes contextos. Habiendo observado "black car" y "blue car" no influenciaba sobre la estimación de la probabilidad de que "red car" apareciera en la secuencia.

Los modelos de redes neuronales son capaces de generalizar mejor. El input de la red neuronal es una secuencia de palabras $w_{1:k}$, y el output es una distribución de probabilidad sobre la próxima palabra. Cada palabra w de entre las k palabras de contexto está asociada a un embedding vector $v(w) \in \mathbb{R}^{d_w}$, y el vector de input x es una concatenación de las k palabras.

$$x = [v(w_1); v(w_2); \ldots; v(w_k)] \tag{4.1}$$

El input x pasa como parámetro de la red neuronal:

$$\begin{aligned} \hat{y} = P(w_i|w_{1:k}) = LM(w_{1:k}) = softmax(hW_2 + b_2) \\ h = g(xW_1 + b_1) \\ x = [v(w_1); v(w_2); \ldots; v(w_k)] \\ v(w) = E_{[w]} \end{aligned}$$

# Capítulo 5

# Elección del lenguaje informático y entorno de ejecución

En este capítulo analizaremos por qué hemos creído conveniente usar Python y otras librerías. En primer lugar, se eligió Python, entre otras cosas, porque vimos que el 57 % de los proyectos de Machine Learning están escritos en Python. Por esto y debido al uso que se le da, tanto a los frameworks como a las librerías relacionadas con este lenguaje, han ido mejorando muchísimo en los últimos años.

Por lo tanto, se pueden usar estructuras de datos y algoritmos relacionados con la inteligencia artificial más facilmente que con otros lenguajes. Entre las librerías destacan: Numpy, que ofrece la capacidad de hacer computación científica y Scipy para computación avanzada. También hemos elegido este lenguaje porque hay muchas fuentes de información en internet para aprender cómo usarlo para inteligencia artificial.

## 5.1. Deep learning frameworks

Para elegir el mejor framework de programación para Deep Learning hemos tenido en cuenta estos cuatro criterios:

- Programación sencilla (desarrollo de software y despliegue ó deployment).
- Velocidad de ejecución.
- Librería open source. Para poder usarla y mejorarla durante el tiempo sin depender de decisiones corporativas externas.
- Información disponible en internet.

Entre los frameworks que hemos estudiado para hacer nuestra elección están:

- Keras

- Pytorch
- TensorFlow
- Theano

Pytorch lo descartamos porque es una librería nueva que solamente se puede usar, por el momento en linux. Además tiene menos tutoriales y papers, de los que extraer información que los demás. Entre las otras tres opciones, nuestra decisión se centró en el primer punto de los cuatro criterios. Keras es una librería de muy alto nivel muy fácil de usar y aprender, además funciona como una API en la que puedes decidir usar Theano o TensorFlow como backend. Hemos aprendido que elegir un lenguaje de tan alto nivel tiene muchos inconvenientes porque para cualquier modificación seria que se quiera hacer de la arquitectura, se debe reescribir el backend y es muy complejo. Por suerte en keras se puede, aunque no lo hayamos tenido que hacer.

## 5.2. Keras

Keras aporta cuatro beneficios claros:

- User friendliness: Minimiza el número de líneas de código que el usuario debe escribir para utilidades muy normales y provee de de un analizador de errores por parte del usuario muy bueno.
- Modularidad: Un modelo, o arquitectura formada entre otras cosas por redes neuronales que se expresan como secuencias o grafos. Contiene módulos completamente configurables que se pueden acoplar muy fácilmente. En particular, las capas de las redes neuronales, funciones de coste, optimizadores, etc. Son todos módulos independientes que se pueden combinar para crear nuevos modelos.
- Fácilmente extensible: Se pueden añadir nuevos módulos como si fueran nuevas clases o funciones. Existen un amplio abanico de modelos que keras provee. Esto es crucial para poder hacer experimentos rápidamente.
- Funciona con Python: Los modelos están escritos en Python y no necesitan archivos de configuración externos para ejecutarlos.

## 5.3. Entorno local de ejecución

Intentamos utilizar la tarjeta gráfica de la universidad, pero no pudimos hacerlo por problemas de temperatura y compatibilidad versiones con otros proyectos, ya que el framework que elegimos no era compatible con la versión de las librerías que requerían otros TFG utilizando la misma tarjeta gráfica, así que tuvimos que recurrir a otros medios como Google Colaboratory[1] o ejecutar los entrenamientos directamente en nuestros ordenadores personales.

[1] https://colab.research.google.com/

Ordenador 1:

- CPU: *Intel*® Core™ i7-6850K
- Placa base: MSI X99A GODLIKE GAMING
- RAM: G.Skill Ripjaws V DDR4 3000MHz 16GB 2x8GB
- Disco duro: Samsung 960 EVO NVMe M.2 SSD PCI-e 500GB
- GPU: Gigabyte GeForce GTX 1050Ti OC 4GB GDDR5

Ordenador 2:

- CPU: AMD RYZEN 7 1800X 4.0GHz
- Placa base: Gigabyte Aorus GA-Z370 Gaming 5
- RAM: G.Skill Ripjaws V DDR4 2666MHz 16GB 2x8GB
- Disco duro: Samsung 960 EVO NVMe M.2 SSD PCI-e 250GB
- GPU: MSI GTX 1080 Gaming X 8GB GDDR5X

## 5.4. Google Colaboratory

Colaboratory es una plataforma de Google en desarrollo, creada para divulgar contenido de investigación y formación sobre el Machine Learning. Es un entorno configurado en Jupyter Notebook que no requiere configuración y que permite ejecutar código en Python de forma gratuita.

Los notebook de Colaboratory se almacenan en Google Drive, y se permite compartirlos como se haría con cualquier otro archivo. Además Google Colaboratory tiene una integración muy sencilla con los documentos de Drive, ya que puede importar y exportar ficheros de forma rápida y con muy pocas líneas de código, mediante una API muy sencilla. También permite ejecutar código de terminal como si se tratara de un entorno Linux para poder tener un seguimiento de los archivos que crea un programa.

La ventaja que tiene usar este entorno de ejecución es que te permite acceder a GPUs que provee Google (GPU Tesla K80). Para resaltar la importancia que ha tenido Colaboratory en nuestro trabajo, no hubiese sido posible ejecutar ni una sola vez nuestro programa en local debido a que nos aparecía un error, en el que se mostraba que nuestras GPUs no tenían recursos suficientes como para terminar la ejecución de un programa computacionalmente tan costoso.

Una particularidad del Google Colaboratory es que solo se puede utilizar una sesión continuada por un máximo de 12 horas, por lo que acabado ese tiempo habrá que reiniciar el proceso. Siempre que se inicia una sesión, se deben realizar este proceso:

- Descargar los archivos de Drive a la sesión de Google Colaboratory.
- Ejecutar código de procesamiento de datos.

- Entrenar el modelo.
- Evaluar el resultado.
- Guardar el resultado.

# Capítulo 6

# Sistema de búsqueda de respuestas médicas

En este capítulo se explicará como hemos usado todo lo aprendido anteriormente para generar respuestas a las preguntas que se le puedan presentar a nuestro sistema. En nuestro caso, como en principio la aplicación debía ser un chat médico, nuestro conjunto de datos es sobre medicina.

## 6.1. El conjunto de datos de entrenamiento

Para nuestro objetivo hemos buscado en la red los training sets que mejor se adaptaran a nuestras necesidades. Después de una amplia y larga búsqueda nos dimos cuenta de que no existen muchos datos dedicados a la búsqueda de repuestas médicas. Encontramos en concreto tres conjuntos de datos de preguntas y respuestas[1] que fueron extraídos de diferentes foros especializados donde el usuario hacía una pregunta a un médico que respondía desde la empresa. Debido, entre otras cosas, a que en esos foros no se cercioraban de hacer un uso correcto del lenguaje tanto gramatical como ortográficamente, decidimos no usarlos. Nuestros directores nos recomendaron leer información acerca de una competición de investigación llamada BIOASQ que premia a los mejores sistemas médicos de entre los que allí compitan. Nos resultó muy útil esa referencia ya que también te ofrecen training sets y test sets con los que poder trabajar.

Dentro de esta competición había dos tareas, de las que solamente una se adecuaba a lo que necesitábamos. Esta actividad examinaba la habilidad de los sistemas para recibir preguntas con conceptos de ontologías médicas y dependiendo del tipo de pregunta, devolver una respuesta exacta y precisa, que en no más que un párrafo responda cualquier pregunta.

### 6.1.1. El conjunto de datos de BIOASQ

El data set, escrito en inglés, ha sido desarrollado por un equipo de expertos

[1] Concretamente los data set son: webmdQA, icliniqQAs y ehealthforumQAs.

biomédicos. Es muy largo y complejo y contiene información de todo tipo, desde texto normal, hasta los links a las fuentes de dicha información. Nosotros, de entre todos estos recursos, usaremos las preguntas (body), los extractos de documentos relacionados con cada pregunta (text de los snippets), y la respuesta exacta (exact answer).

Este data set es un archivo en formato JSON (JavaScript Object Notation). Se compone de un array de "questions", que funcionan como un objeto que contiene la información sobre la misma.

```
{ "questions": [
    {
    "id": "the ID",
    "body": "the question?",
    "type": "the type of the question",
    "concepts": [
        "c1",
        "c2",
        ...
        "cn"
],
"documents": [
    "d1",
    "d2",
    ...
    "dn"
],
"exact_answer": [
    "ea1",
    "ea2",
    ...
],
"ideal_answer": "the ideal answer",
"snippets":[
    {
        "document": "dk",
        "beginSection": "sections. #b",
        "endSection": "sections.#e",
        "offsetInBeginSection": number,
        "offsetInEndSection": number,
        "text": "the snippet"
    }
],
"triples": [
    {
    "o": "object",
    "p": "predicate",
    "s": "subject"
    },
    ...
]
},
...
] }
```

Figura 6.1: Estructura del data set de BIOASQ.

Las preguntas pueden ser de cuatro tipos:

1. Yes/No.
2. Factoid.
3. List.
4. Summary.

Para cada tipo de pregunta, puede haber respuestas "ideal" y "exact", menos para las de tipo "summary" que solo hay respuestas "ideal". Las respuestas "ideal" están restringidas a 200 palabras y en las "exact" cada entidad tendrá un máximo de 100 caracteres. Nosotros utilizaremos las preguntas de tipo "factoid" y "Yes/No" porque nos resultan más sencillas.

**Table 3 Types of questions in Task 1b and respective examples along with the golden answers in each case**

| Question type | Required answer | Example question | Golden exact answer | Golden Ideal answer |
|---|---|---|---|---|
| Yes/No | Exact + Ideal | Is miR-21 related to carcinogenesis? | Yes | Yes. It has been demonstrated in several experimental studies that miR-21 has oncogenic potential, and is significantly disregulated in numerous types of cancer. Therefore, miR-21 is closely related to carcinogenesis. |
| Factoid | Exact + Ideal | Which is the most common disease attributed to malfunction or absence of primary cilia? | "autosomal recessive polycystic kidney disease" | When ciliary function is perturbed, photoreceptors may die, kidney tubules develop cysts, limb digits multiply and brains form improperly. Malformation of primary cilia in the collecting ducts of kidney tubules is accompanied by development of autosomal recessive polycystic kidney disease. |
| List | Exact + Ideal | Which human genes are more commonly related to craniosynostosis? | ["MSX2", "RECQL4", "SOX6", "FGFR1", "FGFR2", "FGFR"] | The genes that are most commonly linked to craniosynostoses are the members of the Fibroblast Growth Factor Receptor family FGFR3 and to a lesser extent FGFR1 and FGFR2. Some variants of the disease have been associated with the triplication of the MSX2 gene and mutations in NELL-1. |
| Summary | Ideal | What is the mechanism of action of abiraterone? | - | Abiraterone acts by inhibiting cytochrome P450 17$\hat{0}$3b1-hydroxylase (CYP17A1), a critical step in androgen biosynthesis, thus leading to inhibition of androgen biosynthesis. |

Figura 6.2: Ejemplos de todos los tipos de preguntas de BioAsq.

## 6.2. Preprocesamiento del lenguaje

Este es el primer paso para generar buenas respuestas, se trata de manejar los datos que obtenemos del data set y cambiarlos para representarlos de tal forma que nuestro modelo sea capaz de procesarlos.

Antes de nada tenemos que abrir el data set y extraer el array de questions, al que nos referimos en la sección anterior. Como habíamos dicho previamente nos quedaremos únicamente con las preguntas, el texto de los snippets, a los que nos referiremos como story, y las respuestas exactas. Estos datos los dividiremos en dos, el 80 % para el training set y el 20 % para el test set.

A todos estos datos, da igual de qué tipo, hay que hacerle una limpieza, con el fin de que sean más acotados y, por lo tanto, más manejables. Esta limpieza normalmente consistiría en modificar el texto con el objetivo de que:

1. Todas las palabras estén en minúscula.
2. Borrar los signos de puntuación de las palabras.
3. Borrar los caracteres no printeables.
4. Borrar números y palabras que contengan números.

Pero en nuestro caso al ser un problema médico, no podemos hacer el punto 2 y 4 porque tenemos tokens como "12 %" o "Emery-Dreifuss" donde mantener tanto los signos de puntuación (entre los que incluimos el % y el guión) como los números es muy importante. Esto nos ha supuesto un verdadero quebradero de cabeza que explicaremos más adelante en el capítulo 7.

A continuación calcularemos la frecuencia con la que aparece cada palabra en el data set, para quedarnos con el 80 % de las palabras más comunes que hay en él, con el fin de empequeñecer los datos y bajar el coste computacional del entrenamiento. Además calcularemos la máxima longitud de entre todas las preguntas, respuestas y story para dar un tamaño a nuestro input, que tiene que ser fijo.

El último paso será dar un id numérico a cada palabra y reescribir todos los datos con estos números, cambiándolos por sus respectivas palabras. A las respuestas les añadiremos las palabras especiales, "start" y "end", al principio y final de la secuencia. No definiremos el input final en esta sección porque nos opupa mucha memoria y excede la capacidad de la memoria RAM en nuestro entorno local (16Gb) y en el Google Colaboratory no nos permiten usar tantos recursos y lanzan un mensaje de error.

## 6.3. El entrenamiento

Hemos tenido que usar una funcionalidad de keras para producir el input final. Consiste en crear una parte pequeña del input para cada iteración del modelo. Nuestro input para cada pregunta se compone de tres vectores que contienen respectivamente la pregunta, la story y el prefijo de la respuesta, además nuestro output será la palabra que sigue al prefijo de la respuesta. Hay que tener un input y output para cada prefijo posible, es por ésto por lo que tenemos múltiples vectores por pregunta, que contienen siempre la misma pregunta y story, y diferentes prefijos de respuesta y output.

| Story | Pregunta | Prefijo de respuesta | Output |
|---|---|---|---|
| While B-type lamins are expressed in almost all cell types, no A-type lamins are present in early vertebrate embryos or undifferentiated embryonal carcinoma cell lines. Intriguingly, expression of A-type lamins occurs concomitant with cell differentiation and embryonic development. | In which cells are A-type lamins expressed? | STARTSEQ | late |
| While B-type lamins are expressed in almost all cell types, no A-type lamins are present in early vertebrate embryos or undifferentiated embryonal carcinoma cell lines. Intriguingly, expression of A-type lamins occurs concomitant with cell differentiation and embryonic development. | In which cells are A-type lamins expressed? | STARTSEQ late | differentiating |
| While B-type lamins are expressed in almost all cell types, no A-type lamins are present in early vertebrate embryos or undifferentiated embryonal carcinoma cell lines. Intriguingly, expression of A-type lamins occurs concomitant with cell differentiation and embryonic development. | In which cells are A-type lamins expressed? | STARTSEQ late differentiating | primary |
| While B-type lamins are expressed in almost all cell types, no A-type lamins are present in early vertebrate embryos or undifferentiated embryonal carcinoma cell lines. Intriguingly, expression of A-type lamins occurs concomitant with cell differentiation and embryonic development. | In which cells are A-type lamins expressed? | STARTSEQ late differentiating primary | cells |
| While B-type lamins are expressed in almost all cell types, no A-type lamins are present in early vertebrate embryos or undifferentiated embryonal carcinoma cell lines. Intriguingly, expression of A-type lamins occurs concomitant with cell differentiation and embryonic development. | In which cells are A-type lamins expressed? | STARTSEQ late differentiating primary cells | ENDSEQ |

Tabla 6.1: Ejemplo de predicción de respuesta usando el modelo de la sección 6.3

Las palabras de la pregunta, la story y el prefijo de respuesta serán codificadas como dense embedding vectors para preservar sus relaciones y propiedades únicas. Mientras que el output al ser una sola palabra la codificaremos usando la técnica del one-hot encoding que nos hará más fácil la tarea de predecir que palabra es la que nos devuelve la red neuronal.

En la siguiente página mostraremos un esquema de nuestro modelo:

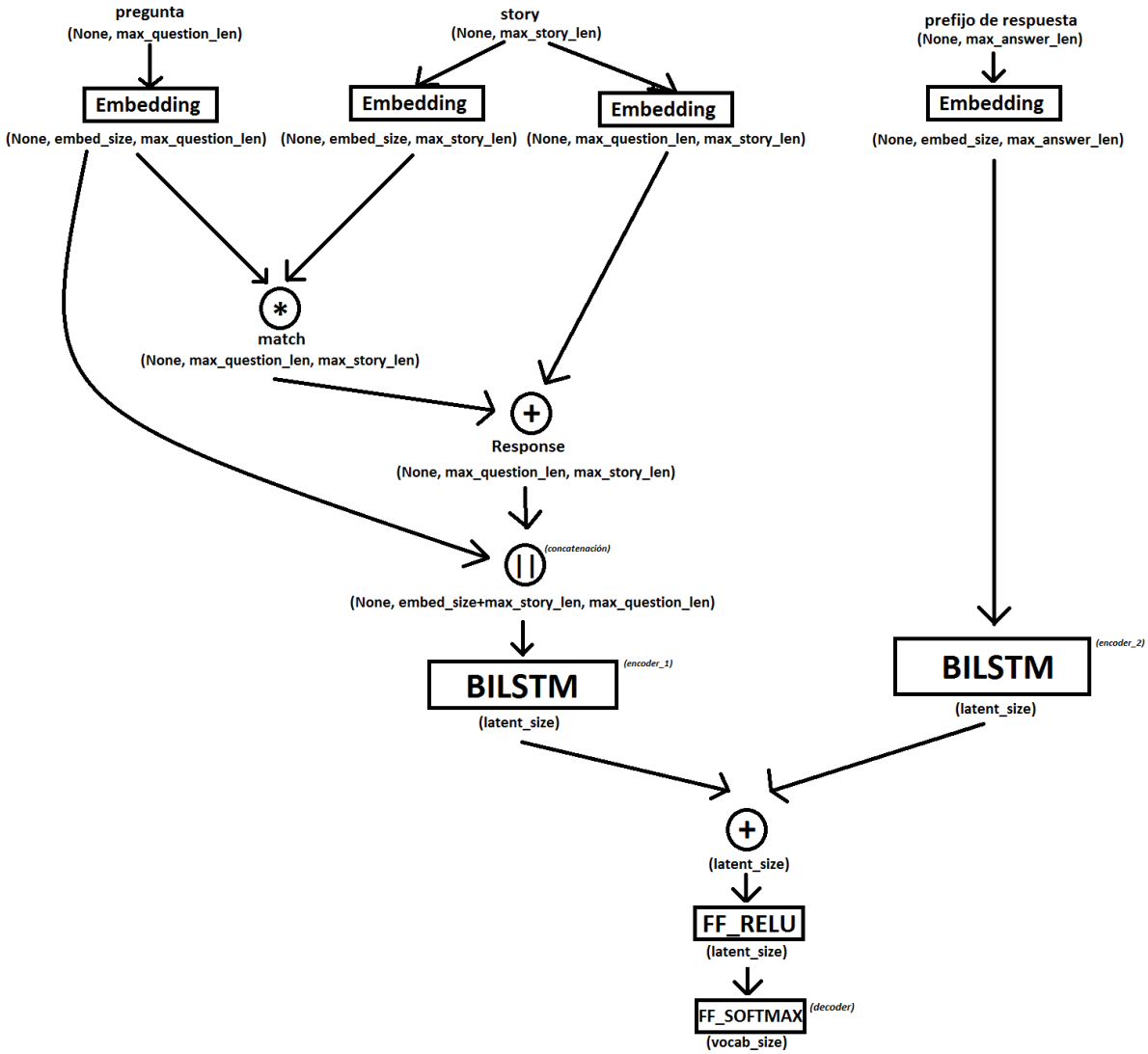


Figura 6.3: Grafo del modelo[2] usado para nuestro entrenamiento.

[2]El modelo está inspirado en los capítulos 7.3 y 7.4 del libro [11] y en los capítulos 7 y 8

Tenemos que volver a la sección 3.6 para poder entender por qué hemos construido este modelo de esta manera. Primero, hay que ver que en general, este modelo es un RNN Transducer (véase 3.6.3), en el que en vez de RNNs simples tiene el modelo representado en la figura 6.3. Cada output de nuestro RNN Transducer será la palabra que va justo después de cada prefijo de respuesta. En el capítulo 7 hablaremos de como hemos combinado sus costes en busca de un mejor resultado.

En cada nodo del RNN Transducer tenemos, en primer lugar, las capas de embedding para cada uno de los 3 vectores del input. Esto sirve para representar los inputs en vectores y poder seguir con la codificación.

El resultado del encoder_1, en la figura 6.3, es con el que más problemas hemos tenido y el más difícil de entender, analizar y encontrar puntos que mejorar. Realmente aquí hemos hecho un proceso de investigación bastante grande en la bibliografía en búsqueda de modelos que pudieran funcionar. Son muy pocos los ejemplos que hay por internet y al final hemos tenido que investigar un poco por nuestra cuenta. En general el "encoder_1" es un Encoder RNN (véase 3.6.2). Pero es más complicado que esto porque al ser búsqueda de respuestas a preguntas en un texto (story) todo se hace más complicado. Deberíamos de ser capaces de extraer la respuesta de la story, sabiendo la pregunta. Adelantamos desde ya que esto último no lo hemos conseguido, hemos tenido resultados que consiguen los objetivos parcialmente pero no es su completitud. Esto lo trataremos en el capítulo 7.

Sabemos por lo explicado en la sección 6.2 que los vectores de input contienen arrays, que son las frases, con ids únicos, que representan las palabras. Una vez pasamos la capa de embedding estos ids se convierten en vectores (véase 4.3) que representan a las palabras. Fíjese en que al input de la story lo pasamos por dos capas diferentes, esto es porque por la naturaleza del problema que nos ocupa, debemos de combinar la pregunta con la story de forma que el ordenador pueda llegar a entenderlo. Para este propósito hemos decidido hacer un producto vectorial de la pregunta y la story, y a eso sumarle la story con los vectores de embedding que tienen el mismo tamaño que el largo máximo de las preguntas (este tamaño se ha escogido por las propiedades de la suma de vectores), y al resultado le volvemos a concatenar las preguntas. Esta última concatenación nos sirve para dar la importancia que merece a la pregunta. Este tipo de codificación del input no se sabe por qué funciona, más allá de por los resultados obtenidos. La mayoría de apartados del Deep Learning se han mejorado, a lo largo de la historia, de manera empírica. Hasta que no haya alguien que descubra por qué funciona un tipo de combinación en vez de otro dependiendo del problema, nosotros no podremos explicar mucho más. Este problema tiene bastante complejidad más allá de que lo que hayamos creado parezca muy simple, hay que pensarlo con mucho detenimiento.

Finalmente este input codificado y modificado, se pasará como parámetro a una BILSTM (véase secciones 3.5, 3.7 y 3.8), que se encargará de representarlo en un vector. Con el BILSTM conseguimos relacionar las palabras en su contexto

del libro [13]

con las palabras que aparecen antes y después en sentidos inversos. Esto nos sirve para crear relaciones más fuertes entre ellas.

A continuación explicaremos el clasificador de nuestro modelo. Para completar el RNN Encoder, necesitamos inputs adicionales, en este caso solamente será uno, que es el prefijo de la respuesta. Este input, obviamente, también tiene que pasar por la capa de embedding para que las palabras sean representadas en vectores y el ordenador sea capaz de relacionarlas. A continuación lo pasamos como parámetro a otro BILSTM.

Estos dos vectores los combinamos por medio de una suma y los pasamos a dos capas ocultas que se encargarán de dar un output con un vector del tamaño del vocabulario en el que cada posición representa el id de la palabra y su valor es la probabilidad de que esa palabra sea la siguiente en aparecer. Esta probabilidad la conseguimos gracias a las funciones de activación RELU junto a la Softmax que devuelve valores en el mismo rango que la Sigmoide (véase 3.1.2).

## 6.4. Predicción y evaluación

### 6.4.1. Predicción

Para hacer la predicción cogeremos como ejemplo la primera pregunta, story y respuesta del dataset. El primer prefijo de respuesta que pasaremos como parámetro a nuestro modelo será el caracter especial "start". A partir de ahí iremos añadiendo cada una de las palabras predichas y se las añadiremos al prefijo de respuesta. Repetiremos el proceso pasándole siempre como parámetros al modelo el mismo story, pregunta y el prefijo actualizado. Pararemos si el modelo predice el caracter especial "end" o alcanzamos un número de palabras igual a max_answer_len.

### 6.4.2. Evaluación

Usaremos la librería BLEU de NLTK (Natural Language Toolkit) para evaluar la calidad de las respuestas dadas por nuestro modelo. Esta librería se usa sobre todo para evaluar traducciones.

Nosotros estamos interesados únicamente el la funcionalidad Individual n-gram Scores. Un n-gram es una secuencia compuesta por n palabras. Esta funcionalidad nos permite calcular cuántos 1-gram, 2-gram, 3-gram y 4-gram coinciden en la respuesta dada por el modelo y la respuesta del data set. También podemos calcular la evaluación combinando los cuatro n-gram. Concretamente nosotros hacemos cuatro evaluaciones para hacer un análisis más completo de nuestras respuestas:

1. Cuántos 1-gram iguales hay.
2. 50 % de importancia a los 1-gram y 2-gram.
3. 30 % de importancia a los 1-gram, 2-gram y 3-gram.

4. 25 % de importancia a los 1-gram, 2-gram, 3-gram y 4-gram.

# Capítulo 7

# Experimentos y resultados

El primer experimento que hicimos para saber si nuestro modelo funcionaba fue usar un data set mucho más sencillo, que es el que se recomienda en la bibliografía para empezar. Este data set también se compone de la tripleta story, pregunta y respuesta, pero son datos muy sencillos.

```
1 Mary moved to the bathroom.
2 John went to the hallway.
3 Where is Mary?        bathroom        1
4 Daniel went back to the hallway.
5 Sandra moved to the garden.
6 Where is Daniel?      hallway 4
7 John moved to the office.
8 Sandra journeyed to the bathroom.
9 Where is Daniel?      hallway 4
10 Mary moved to the hallway.
11 Daniel travelled to the office.
12 Where is Daniel?     office  11
13 John went back to the garden.
14 John moved to the bedroom.
15 Where is Sandra?     bathroom        8
1 Sandra travelled to the office.
2 Sandra went to the bathroom.
3 Where is Sandra?      bathroom        2
```

Figura 7.1: Pequeño extracto de uno de los data sets de Facebook's Babl tasks.

Con este data set tan sencillo no nos costó nada más que, ejecutar más tiempo el entrenamiento con un tamaño de batch menor (reducir el batch significa no ejecutar la función de optimización hasta entrenar un número de ejemplos igual a batch), para conseguir más del 90 % de acierto en los datos de test.

El siguiente paso era hacer que funcionara con el data set de BioAsq. El principal inconveniente era, que en el antiguo modelo solamente había respuestas con una sola palabra. Se nos ocurrió primero, dada la dificultad, hacer un clasificador de respuestas donde el ordenador no generaba respuestas desde cero sino que elegía de entre las respuestas que tenía el data set (retrieval-based model), la que consideraba que era la que respondía a la pregunta. Este modelo no nos servía, porque pretendíamos que el ordenador aprendiera a responder, no a elegir una

respuesta (generative-based model).

Aquí fue donde hicimos el modelo explicado en la sección 6.3. Hemos hecho pruebas con diferentes funciones de optimización, como el stochastic gradient descent, Adam o RMSprop, que son funciones que trae por defecto Keras. Además como habíamos representado el output con un one-hot encoding (véase 4.3.1) nos sirve una función de coste que se llama categorical_crossentropy que Keras ha diseñado específicamente para este tipo de codificación. Aún así hemos probado otras, como por ejemplo, el mean_squared_error que también se podía intuir por la forma de la función que podía ayudar a conseguir el resultado más óptimo, aunque es muy ondulante.

Lo primero que hicimos fue probarlo para 10 elementos del training set y ver si funcionaba. Después de 500 iteraciones completas del entrenamiento y más o menos 1 hora de reloj, conseguimos unos resultados tan buenos como los siguientes.

| Pregunta | Respuesta predicha | Respuesta correcta |
|---|---|---|
| What is the substrate of the microbial enzyme inulinase? | - - in in mhc and and - fructose fructose | The inulinase acts on the beta-(2,1)-D-fructoside links in inulin releasing D-fructose. |
| What tyrosine kinase, involved in a Philadelphia- chromosome positive chronic myelogenous leukemia, is the target of Imatinib (Gleevec)? | bcr - abl | BCR-ABL |
| When was empagliflozin FDA approved? | sushi . r | Sushi.R |
| Which R/bioconductor package is used for integrative genomics visualizations? | 2014 | 2014 |
| How many genes are imprinted in the human genome? | | fewer than 100 |
| In which cells are A-type lamins expressed? | late differentiating primary cells | late differentiating primary cells |
| Gene silencing can be achieved by RNA interference (RNAi) in eukaryotic organisms. What is the name of the analogous process in prokaryotic organisms? | crispr - cas | CRISPR-Cas |
| How is connected ïsolated Noncompaction cardiomyopathyẅith dilated cardiomyopathy? | via mutations in beta beta - alpha - tpm1 | via mutations in beta-MHC and alpha-TPM1 |
| What is the rate of survival after commotio cordis? | 10 - 28 % | 10-28 % |

Tabla 7.1: Resultados al evaluar el training set de nuestro sistema de búsqueda de respuestas.

Ahora bien, como podemos observar, el neural language model (véase 4.4) funciona bien. Además vemos que sabe diferenciar las preguntas y acierta bastante a la hora de dar la respuesta. Esto demuestra que hemos sabido hacer los dos primeros puntos de los objetivos para hacer un buen IA chatbot de la sección 1.2.1.

El incoveniente de este modelo que hemos creado es que no generaliza y al hacer la prueba con el test set completo, vemos que es incapaz de extraer las respuestas de las story y da respuestas sin sentido. Aún así, creemos que la importancia de este error reside en el modelo en sí, en su estructura, debido a que como hemos comentado anteriormente el vocabulario médico es muy particular y tiene palabras que el tokenizador de Spacy no es capaz de ver como una sola y convierte el vocabulario en algo extraño de manejar. Esto lo podemos observar en los espacios de los guiones en las respuestas predichas en el cuadro 7.1.

Decidimos cambiar la representación del output de un one-hot encoding a po-

nerle a cada id de palabra un valor que representaba la relación de dicha palabra con la palabra del output. Esta es una de las muchas funciones que tiene Spacy, que es una gran librería. Así, solo nos quedaba cambiar las funciones de coste por otras que nos vinieran mejor. Como por ejemplo, el cosine_proximity que da un valor a la similitud que hay entre dos vectores, es decir el output predicho y el real. Es este último el que tendría esta representación y el otro es sobre el que se ejecutan las funciones de optimización.

Tenía todo el sentido del mundo pero otra vez aprendimos que la calidad de los resultados hay que medirlos de forma empírica, pues fue mucho peor que con la representación anterior. De ahí que finalmente hayamos concluido que como en realidad este TFG es un trabajo de investigación lo que importa no son los resultados sino lo que uno puede concluir de ellos.

Aún así hemos utilizado el modelo de clasificación visto anteriormente con múltiples cambios para intentar solucionar solamente las preguntas de tipo “Yes/No”.

El modelo es el siguiente:

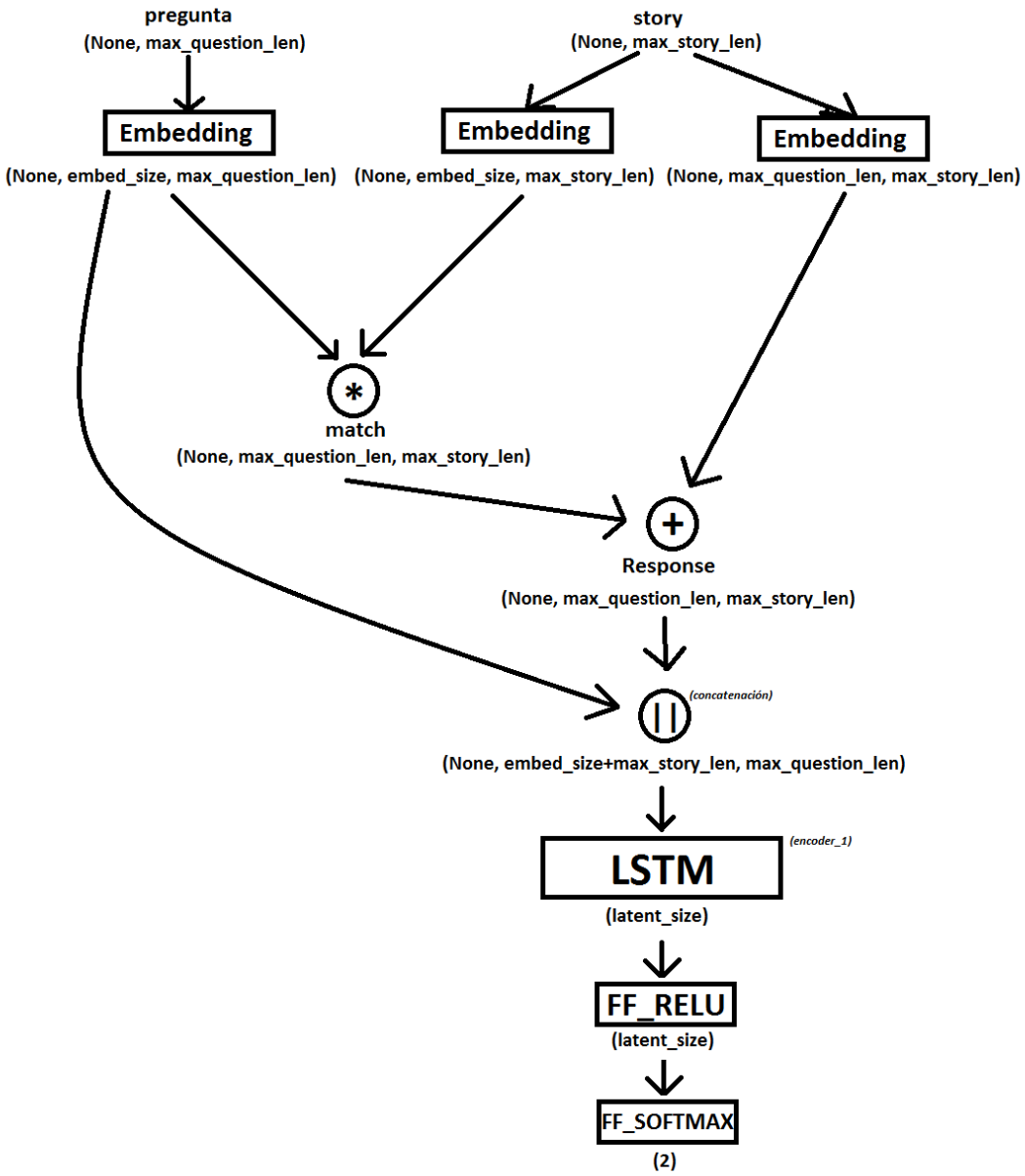


Figura 7.2: Grafo que describe nuestro modelo de clasificación.

Como podemos apreciar nuestro modelo ha pasado de ser un RNN Transducer con un RNN Encoder dentro de cada nodo a ser un RNN Acceptator (véase 3.6.1) dentro de cada nodo del RNN Transducer.

Este modelo, dados una story y una pregunta produce un número entre cero y uno que representa la probabilidad de que la respuesta a la pregunta sea "yes". Bastaría pues con encontrar la frontera de decisión (véase 3.1.2) que nos separe las respuestas predichas como "yes" de las respuestas predichas como "no", sin embargo esto no resulta tan trivial, ya que las predicciones para preguntas que deberían haber sido respondidas con un sí están entremezcladas con las que deberían haber sido respondidas como no.

El principal problema es la falta de ejemplos negativos en el conjunto de datos (más o menos una proporción del 8 %), lo cual nos produce unos modelos que, prediciendo todas las respuestas a "yes" (independientemente de la pregunta) consiguen una precisión en el conjunto de test (que guarda la misma relación entre "yes/no" que el conjunto de entrenamiento) de un 92 %, lo que en un principio puede parecer un resultado favorable, pero no lo es.

Ajustando la frontera de decisión conseguimos que nos predijese correctamente una respuesta como "no", lo que nos llevó a una precisión del 93 %, pero de ahí no hemos sido capaces de subir. Hicimos distintas pruebas cambiando las funciones de coste (Binary Crossentropy, Mean Squared Error, Hinge...) y las funciones de optimización (RMSprop, Adam, Adadelta, Stochastic Gradient Descent...), disminuyendo el tamaño del batch, e incluso replanteamos el problema para que fuese tratado como one-hot encoding, es decir, tratar las respuestas del conjunto de datos y las salidas como vectores bidimensionales en los que [1,0] indica "yes" y [0,1] indica "no" (con one-hot encoding representamos cada palabra del vocabulario con un vector de ceros con un uno en la posición del vocabulario que ocupa la palabra, es decir el número de dimensiones del vector es el tamaño del vocabulario pero en este caso el vocabulario está formado exclusivamente por "yes" y "no"). Con esta codificación probamos a utilizar la función de coste Categorical Crossentropy pero no produjo ninguna mejora a los resultados anteriores.

# Capítulo 8

# Conclusiones y trabajo futuro

Nos gustaría pensar que los resultados de nuestro sistema de preguntas y respuestas son parte de una investigación más grande y que se pueden llegar a alcanzar buenos resultados. Aún así, somos conscientes de que nuestros resultados pueden mejorar mucho. En este capítulo explicaremos desde nuestro punto de vista de por qué está funcionando mal y como lo arreglaríamos si hubiesemos tenido más tiempo.

En primer lugar nos gustaría recalcar que el concurso que planteaba BioAsq no es un concurso para estudiantes sino para equipos más serios de investigación. Que normalmete se componen de tres grupos de trabajos especializados respectivamente en: el preprocesamiento, el entrenamiento y en la evaluación. Esto lo hemos podido comprobar en los diversos artículos científicos que hemos leído (véase [19] o [14]).

En mi opinión nuestro principal problema fueron las story. En el conjunto de datos podemos observar como hay palabras muy complejas que la librería Spacy no podía reconocer bien. En su lugar hemos visto como hay equipos de investigación que usan un sistema de Named Entity Recognition para reconocer este tipo de palabras o términos. En concreto hemos descubierto que existe una API[1] que nos permitiría hacerlo más fácilmente. Aunque seguiría siendo complejo y no podemos asegurar que funcione al 100 %.

En segundo lugar dentro de la problemática de las story, es que hay muchas por cada pregunta. Nosotros lo hemos solucionando concatenándolas creando una sola story grande. Seguramente existan soluciones mucho mejores, pero que nosotros no hayamos caído en la cuenta.

Si nos vamos a soluciones más trabajadas y complicadas podemos observar que hay posibles modificaciones de nuestro modelo que podrían haber aumentado su efectividad. En concreto un sistema de atención[2] sirve para comunicarle a

[1] `https://willieboag.files.wordpress.com/2018/03/clineramiacri2015_poster.pdf`
[2] Explicado en el capítulo 17.5 del libro [12].

nuestro Decoder en qué partes del vector de output del Encoder prestar más atención según la pregunta. Hemos decidido no explicar esto en la memoria debido a que no lo sabemos implementar a nivel práctico y se quedaba fuera de los límites de nuestro TFG. De todas maneras, en nuestra opinión, encajaría como una pieza fundamental más del puzzle.

En la parte de la predicción también deberíamos de hacer muchos cambios pues nosotros damos por buena la palabra más probable que recibimos del vector de output del Decoder. En nuestro caso ha tenido que ser así debido a que términos como "BCR-ABL" o ponderaciones como "10-28 %", es muy importante, incluso crucial, que sean esas y no otras las que salgan como respuestas. Aún así existen métodos como el Beam Search Decoder[3] que sería muy interesante observar como funcionan. Aún así y dada la complejidad de las palabras y términos de nuestros datos no nos arriesgaríamos a asegurar que fuera la mejor opción. Sin embargo, son muchos los sistemas de búsqueda de respuestas, sistemas de traducción, y un largo etcétera, los que han demostrado más de una vez su efectividad.

Con respecto al modelo de clasificación el problema que creemos que causaba el mal entrenamiento era la falta de ejemplos negativos en el conjunto de datos de entrenamiento. Una posible solución sería, tal y como aprendimos con el curso [1] insertar falsos negativos en el data set, es decir aumentar la proporción de negativos (modificando preguntas o incluyendo nuevas) para que el modelo no obtenga una alta precisión prediciendo todo a sí.

Para finalizar, esperamos que les haya resultada interesante la lectura y, por qué no, hayan aprendido algo nuevo. Y les queremos agradecer la atención depositada en cada página, ya que no son conceptos fáciles de leer y entender.

[3] `https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/`

# Capítulo 9

# Conclusions and future work

We would like to think that our question-answer system's results are part from a bigger investigation and that good results can be achieved. Even though, we are conscious that our results can improve severely. In this chapter we will explain in our opinion why it is getting worng results and how we would fix it if we would have had more time.

First of all we would like to recall that BioASQ's contest wasn't intended for students but for more serious investigation teams, that normally are made of three specialized workgroups in preprocessing, training and evaluation. This we have been able to check in diverse scientific articles we have read (note [19] or [14]).

In our opinion our principal problem were the stories. In the data set we could observe how there are very complex words that the Spacy library could not recognize properly. Instead we have observed that there are investigation teams using a Named Entity Recognition system for recognizing this kind of words or terms. Specifically we found an API that would let us perform this task more easily, even it would still be complex and we cannot assure that it would work 100 %.

Another problem regarding the stories is that there are a lot for each question. We partially solved this concatenating them such as there is only one final big story. Surely there are better solutions but we haven't noticed.

If we go to more worked and complex solutions we can observe that there are possible modifications of our model that could have raised its efectivity. Precisely an attention mechanism[12] is useful for communicating to our Decoder in which parts of the Encoder's output vector it should pay more attention according to the question. We have decided not to explain this in the memory because we don't know how to implement this in a practical level and it was beyond the scope of our Final Grade Project. In any case, in our opinion, it would fit as another fundamental piece of the puzzle.

We should also do a lot of changes in the prediction part, because we accept the most probable word that we recieve from the Decoder's output vector. In our case it had to that way because terms such as "BDR-ABL" or weighings such as "10-28 %", it is very important, even crucial that those and not others are the answers. Also there are methods such as Beam Search Decoder[1] that it would have been very interesting to see how they work. Even though and given the complexity of our data's words and terms we could not risk to assure that it is the best option. Nevertheless there are a lot of answer retrieval systems, translation systems, etc. that have proved more than once its efectivity.

In regards to the classification model, the problem we think it was causing the bas training was the lack of negative examples in the training set. One possible solution was, as we learnt in the course [1], to insert fake negatives in the data set, that is, raising the proportion of negatives (modifing questions or adding new ones) so the model does not get a high accuray predicting all to yes.

Concluding, we hope this lecture has been interesting and, why not, didactic. We would like to thank the attention given to each page because this are not easy to read or understand concepts.

[1] `https://machinelearningmastery.com/beam-search-decoder-natural-language-processing/`

# Capítulo 10

# Trabajo personal

## 10.1. Daniel Reyes Parrilla

En primer lugar, llevamos trabajando en este proyecto desde el segundo cuatrimenstre del curso 2016/2017. Nuestro director, Alberto, por aquel entonces era mi profesor de IA y propuso este TFG para el que quisiera participar. Al haber más candidatos de los esperados, nuestro director decidió evaluarnos mediante un estudio escrito sobre los chatbots en general. Yo realicé ese trabajo con ayuda de mi compañero Ignacio nutriéndonos de fuentes de Internet.

Antes de comenzar el verano, nuestros directores, nos facilitaron la memoria del TFG del alumno del Doble Grado en Ingeniería Informática y Matemáticas, Ángel Javier Alonso Hernández (véase [20]). Este documento, nos sirvió más que para comprender lo que este alumno había hecho, para darnos cuenta del desafío que teníamos por delante.

A partir del comienzo de las clases después del verano, me inscribí en el curso [1] para aprender los fundamentos del Machine Learning[1]. Este curso tenía una base teórica muy fuerte que se acompañaba por una parte práctica programada en Octave (del cual no tenía conocimientos previos).

Al finalizar este curso, a mediados del mes de febrero, nos dimos cuenta por fin de qué era lo que teníamos que hacer y vimos que no era nada trivial. El problema de sistemas de respuestas, muchas veces no salen en los libros de Deep Learning en NLP dada su dificultad, y en los que lo hacen salen en las últimas páginas. Por consiguiente, me puse rápidamente manos a la obra, y empecé a leer el libro [12] obviando los temas iniciales que trataban los mismos conceptos que en el curso [1] estudiado anteriormente.

Al mismo tiempo, estudié cursos online sobre Python, ya que era un lenguaje de programación que nunca había usado. También a la vez, estudié junto a Ignacio los posibles frameworks de Deep Learning, y escogimos la idea de hacerlo en

[1]Se puede acceder a mi diploma del curso mediante el link: `https://www.coursera.org/account/accomplishments/certificate/U3LRLR24ADTR` y al programa completo en: `https://www.coursera.org/learn/machine-learning`

Keras.

Cuando terminé el libro fue cuando me di cuenta de que, aún así, no había empezado a entender bien los procesos que tenía que seguir en nuestra implementación para poder hacer aprender al ordenador a generar respuestas a partir de preguntas y story. Nuestros directores, para ayudarnos, nos facilitaron acceso a un servidor de la universidad (holstein) y al código de Ángel (alumno del que obtuvimos la memoria). Ejecutar el código no fue sencillo, ya que nos encontramos con problemas de versiones y con que el data set no venía incluido en la entrega de Ángel. Ayudé a Ignacio a configurarlo todo y a intentar entender el código.

Finalmente decidimos no utilizar este código porque era de muy bajo nivel, difícil de entender y por consiguiente muy difícil de adaptar a nuestro TFG. Fue entonces cuando empecé a leer el libro [11] y entender bien como es la implementación de las teorías del Deep Learning en Keras. Al final de mi lectura de este libro fue cuando implementé junto a mi compañero el primer y segundo experimento de la sección 7. Para esta implementación configuré el framework Keras con todas sus dependencias con el programa Anaconda en mi ordenador personal.

Aún así no sabíamos como generar respuestas. Leyendo por Internet accedí a un blog[2] en el que se publicitaba un libro ([13]) con información de la implementación de un sistema de generación de descripciones de imágenes. Nos servía porque para implementar el neural language model (4.4), da igual que tipo de información se codifique: ya sea música, imágenes o texto. Compré ese libro entendí como se contruía su sistema y diseñé el modelo de este TFG partiendo desde cero.

Para poder ejecutar esta última versión del TFG, nos hizo falta aprender como usar Google Colaboratory, sacamos información muy válida de la memoria de TFG [21].

Para finalizar, participé activamente en todos los experimentos que se hicieron, teniendo una mayor importancia en el modelo de las preguntas de tipo "Factoid", pero siempre con ayuda de mi compañero Ignacio. Además escribí más o menos la mitad de la escritura de la memoria, pero aportando más en los últimos capítulos.

[2]https://machinelearningmastery.com/

## 10.2. Ignacio Terriza Díez

Al igual que mi compañero Daniel, comencé a investigar acerca del estado del arte de los chatbots antes de comenzar el curso 2017/2018, ya que cuando me propuso realizar el Trabajo de Fin de Grado sobre este tema me pareció bastante interesante.

Cuando fuimos seleccionados para este proyecto nuestro director Alberto nos pasó una copia de la memoria de otro alumno que el año anterior se aventuró con otro proyecto de Deep Learning, Ángel Javier Alonso Hernández. La primera vez que leí la memoria no tuve mucho éxito entendiendo los conceptos que ahí se trataban, pero poco a poco durante el curso tras las distintas fases de investigación cada vez que releía la memoria entendía un poco más hasta llegar a comprenderla por completo.

La primera recomendación de nuestros directores fue que nos apuntásemos a un curso gratuito de la plataforma Coursera impartido por Andrew Ng.[3] en el que abordaba los principios del Machine Learning, además de ofrecer una serie de ejercicios evaluables sobre los conceptos aprendidos. Estos ejercicios se habían de implementar en Octave, un lenguaje de programación especializado en realizar cálculos numéricos muy similar a MATLAB.

Tras terminar el curso se nos abrían muchas vías por donde seguir. Andrew Ng ofrecía toda una serie de cursos especializados en Deep Learning, pero esta vez ya no eran gratuitos. Pudimos acceder al contenido teórico de forma gratuita, pero sin el soporte ni la evaluación de ejercicios.

Después de esto tocaba ponerse manos a la obra. Lo primero que intentamos fue ejecutar el código de Ángel en el servidor ofrecido por la universidad (holstein). El trabajo de Ángel estaba enfocado en el resumen de textos adaptando un modelo de traducción existente, e incluía un data set bastante grande. Además, para el entrenamiento requería cerca de cuatro días. Ésto nos causó serios problemas ya que nunca llegamos a poder terminar de ejecutar el entrenamiento, debido a que el servidor siempre se caía cuando llevaba un tiempo entrenando. Más tarde llegó a nuestro conocimiento que había más alumnos utilizando el servidor para entrenar modelos de Deep Learning, pero utilizaban un framework diferente llamado PyTorch. El problema era que PyTorch requiere de la versión de la librería CUDA (para computaciones con la tarjeta gráfica) 9.1, y sin embargo Keras no funciona más allá de la 9.0.

Habiendo descubierto la causa del problema decidimos ejecutar el entrenamiento en otro lado. Había varias alternativas como Google Collaboratory, Watson Studio de IBM o incluso entrenar el modelo en nuestros ordenadores personales. Pero para estas alturas mi compañero Daniel ya había comenzado a desarrollar un primer modelo más adaptado a nuestro problema, y dado que el tiempo de entrenamiento era mucho más corto nos centramos directamente en este nuevo modelo.

[3]Se puede acceder a mi diploma mediante `https://www.coursera.org/account/accomplishments/certificate/MPFE6A747FUA`

Recomendado por mi compañero Daniel comencé a leer el libro [11] para familiarizarme con el framework que íbamos a utilizar, y juntos comenzamos a programar y probar distintos modelos. He de decir que las ideas generales de los modelos que íbamos desarrollando eran de mi compañero ya que ha profundizado más en el tema que yo.

Ambos participantes hemos tomado parte en el desarrollo y ejecución de los experimentos, quizás yo me he especializado más en el modelo de clasificación de preguntas del tipo sí o no.

Las competencias adquiridas durante el desarrollo de este Trabajo de Fin de Grado incluyen, además de conocimientos teóricos acerca del Deep Learning y prácticos sobre los frameworks para su desarrollo (Keras principalmente), aprendizaje del lenguaje Octave (requerido para el curso de Andrew Ng.), aprendizaje de Python mediante varios cursos adquiridos online (necesario para utilizar Keras) y gestión de distribuciones de Python mediante la herramienta Anaconda[4], que permite crear entornos virtuales con las versiones específicas de los paquetes de Python que se necesiten.

[4] `https://www.anaconda.com/`

# Bibliografía


[1] Andrew Ng, *Machine Learning*, Recuperado de `https://www.coursera.org/learn/machine-learning#`, 2011.

[2] S. Raval, *How to Make a Text Summarizer*, Recuperado de `https://www.youtube.com/watch?v=ogrJaOIuBx4`, 2017.

[3] Maruti Techlabs, *What Are The Best Intelligent Chatbots or AI Chatbots Available Online?*, Recuperado de `https://chatbotsmagazine.com/which-are-the-best-intelligent-chatbots-or-ai-chatbots-available-online-cc49c0f3569d`, 2017.

[4] Maruti Techlabs, *HOW TO MAKE AN INTELLIGENT CHATBOT?*, Recuperado de `https://www.marutitech.com/make-intelligent-chatbot/`, 2017.

[5] HealthTap Inc, *Dr. A.I. by HealthTap*, Recuperado de `https://www.amazon.com/HealthTap-Inc-Dr-A-I-by/dp/B06WRSVQH9`, 2017.

[6] Tsatsaronis, G; Balikas, G; Malakasiotis, P; Partalas, T; Zschunke, M; Alvers, Michael R; Weissenborn, D; Krithara, A; Petridis, S; Polychronopoulos, D; Almirantis, Y; Pavlopoulos, J; Baskiotis, N; Gallinari, P; Artiéres, T; Ngonga,A; Heino, N; Gaussier, E; Barrio-Alvers, L; Schroeder, M; Androutsopoulos, I; Paliouras, G., *Overview of the BIOASQ large-scale biomedical semantic indexing and question answering competition* Recuperado de `https://bmcbioinformatics.biomedcentral.com/track/pdf/10.1186/s12859-015-0564-6?site=bmcbioinformatics.biomedcentral.com`, 2015.

[7] Binny, M; Aasim, O., *Principal Component Analysis Tutorial*, Recuperado de `https://www.dezyre.com/data-science-in-python-tutorial/principal-component-analysis-tutorial`, 2018.

[8] Velasco, F., *Introduction to Deep Learning Part 3: Recurrent neural networks & LSTM*, Recuperado de `http://www.stratio.com/blog/deep-learning-3-recurrent-neural-networks-lstm/`, 2017.

[9] Spacy, *Linguistic Features*, Recuperado de `https://spacy.io/usage/linguistic-features`, 2017.

[10] Google, *Hello, Colaboratory*, Recuperado de `https://colab.research.google.com/notebooks/welcome.ipynb`, 2017.

[11] Gulli, A; Pal, S., *Deep Learning with Keras*: Packt Publishing, 2017.

[12] Goldberg, Y., *Neural Network Methods for Natural Language Processing*: Morgan & Claypool publishers, 2017.

[13] Brownlee, J., *Deep Learning for Natural Language Processing*: Jason Brownlee, 2018.

[14] Wiese, G; Weissenborn, D; Neves, M., Neural Question Answering at BioASQ 5B, 2017.

[15] Pathak, P; Goswami, R; Gautam, J; Patel, P; Patel, A., *CRF-based Clinical Named Entity Recognition using clinical NLP Features*, 2010.

[16] Choi, Y; Yi-I, C; Sontag, D., *Learning Low-Dimensional Representations of Medical Concepts*, 2015.

[17] Sutskever, I; Vinyals, O; V.Le, Q., *Sequence to Sequence Learning with Neural Networks*, 2014.

[18] Kingma, D; Lei, J., *ADAM: a Method for Stochastic Optimization*, 2015.

[19] Mollá, D, *Query-based Summarisation Techniques for Selecting the Ideal Answers*, 2017.

[20] Alonso Hernández, A, *Deep Learning aplicado al resumen de textos*, 2017.

[21] Díaz Badra, R, *Clasificador de Vestimenta basado en redes neuronales*, 2018.

# Apéndice A

# Ejecución del modelo de generación de respuestas

En este apéndice se explicará como es el proceso de ejecución del sistema de preguntas y respuestas basado en Deep Learning del tipo de preguntas "Factoid".

Lo primero que hay que hacer es crear una cuenta en `https://colab.research.google.com`. Una vez creada, abrimos un cuaderno de Python 3.

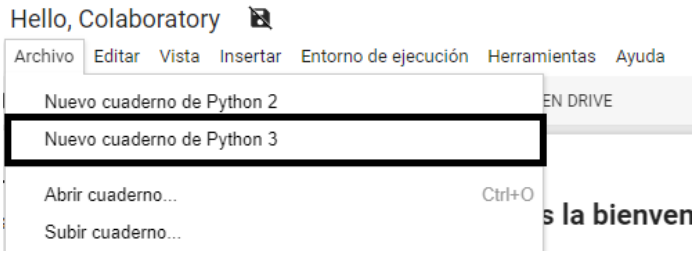


Figura A.1: Captura del proceso de creación de un cuaderno de Python 3 en Google Colaboratory.

A continuación, elegimos la GPU como entorno de ejecución.

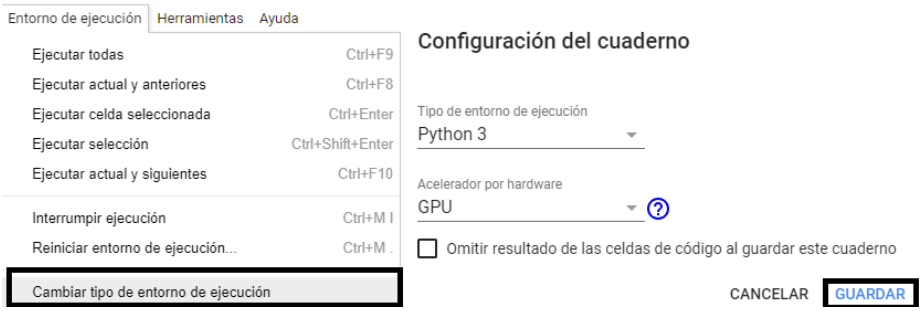


Figura A.2: Captura del proceso de creación de configuración del entorno de ejecución.

Una vez hemos creado y configurado el cuaderno, se ha de copiar y pegar en el cuaderno el contenido de "colab.py" que hay en Github. Este archivo contiene las funciones necesarias para descargar en el entorno de ejecución los archivos del Google Drive y a cargar en el mismo los archivos que hayamos creado durante el proceso de ejecución. Parar ejecutar el código pegado hay que pulsar SHIFT+ENTER. A continuación nos aparecerá un enlace de autenticación. Se debe acceder a él, seguir los pasos que se muestran y copiar el código de autenticación para después pegarlo en la celda que aparece en el cuaderno.

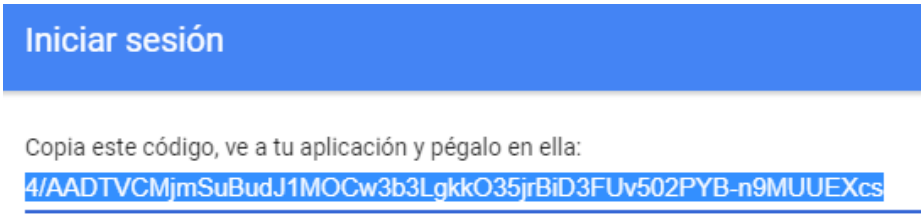


Figura A.3: Captura del proceso de autenticación en el cuaderno de Google Colaboratory.

A continuación, subimos el data set "BioASQ-trainingDataset5b.json" a la carpeta principal del Drive y ejecutamos el método seeFiles(). Nos aparecerá algo parecido a esto:

```
title: deep_learning_for_nlp.pdf, id: 1ktmbjp3WYLoFLSZRbXRuX8rpqz2eQJ0U
title: model.h5, id: 1fCBycEMQChV6nPo58N6ObbGR7wYfbsnc
title: input_test.pickle, id: 1CTMQnH77R5Wv4actNfwcUip3Z5jlgbU_
title: dataTest.pickle, id: 1ZSFjUvEP2WtJ2_gRx66ZupsUyU2ye9a6
title: input_train.pickle, id: 1Qk8v3t7_vFGrVXaUXx5LFVm_nuZtn3ps
title: dataTrain.pickle, id: 1SvcbGiasv0QWiNa72l9hIC5KGnraxY99
title: id2word.pickle, id: 1AWzpgQVtfLZZd9EyM5VLxBIqqRr-TDKU
title: BioASQ-trainingDataset5b.json, id: 1yxyKCeDmJuHqiCYIogI2bj19_sI_D6Rc
title: Documento sin título, id: 1E0CPCm4aJoXZvWEpz9o-uWt5OEZCTS_e-KRQrT0QB8g
title: Documento sin título, id: 1aoExowmLGGmpr6UH1sv3zz-c5xMfAhRyefL2jYPir3g
title: main.py, id: 0B6kLn5jfgF7iTHl3dmtWUEQ4dVpwZ3RfRDRwZndGQ2RrS2l3
```

Figura A.4: Ejemplo de ejecución del método seeFiles().

Una vez que nos sale el data set al ejecutar este método, debemos modificar el método downloadFiles() cambiando la id que hay puesta por la que sale en la lista.

```
def downloadFiles():
  downloadFile('1yxyKCeDmJuHqiCYIogI2bj19_sI_D6Rc', "BioASQ-trainingDataset5b.json")
```

Figura A.5: Modificación del método downloadFiles().

Para cuando esté hecho, ejecutamos el método:

downloadFiles()

Lo siguiente que hay que hacer es instalar el paquete “en_core_web_md” de Spacy ejecutando el siguiente comando:

!pip install https://github.com/explosion/spacy-models/releases/download/en_core_web_md-2.0.0/en_core_web_md-2.0.0.tar.gz

Ahora ya estamos preparados para ejecutar el preprocesamiento (“sentenceEmbeddings.py”), el entrenamiento (“main.py”) y la evaluación (“evaluate.py”). Deberemos copiar y pegar el contenido de los tres archivos de la misma manera que hicimos con el archivo “colab.py”.

Para finalizar, si es menester, se puede guardar el modelo en Google Drive ejecutando el método createFile() de la siguiente manera:

createFile(’model.h5’)

El código para ambos modelos se encuentra en el repositorio `https://github.com/nachoterriza/Sistema-de-busqueda-de-respuestas-con-Deep-Learning`.

# Apéndice B

# Ejecución del modelo de clasificación

En este apéndice se explicará como es el proceso de ejecución del sistema de preguntas y respuestas basado en Deep Learning del tipo de preguntas "yesno".

Este modelo, debido a su menor complejidad y tiempo de entrenamiento, lo hemos ejecutado siempre en nuestros ordenadores personales, por ello explicaremos como montar el entorno y ejecutar el entrenamiento en local.

Primero de todo vamos a utilizar Anaconda para gestionar las versiones de Python. Sobre Anaconda crearemos un nuevo entorno, en el que instalaremos los paquetes de Python necesarios, incluyendo Keras.

Las librerías necesarias para el uso de la GPU son las siguientes:

- CUDA v9.0 - el paquete CUDA de NVIDIA.
- CUDNN v7.0 - librerías adicionales de NVIDIA para CUDA para programar redes neuronales.

Es muy importante la versión de las librerías ya que Keras no funciona con versiones posteriores.

Una vez instaladas las librerías de la GPU comenzamos a configurar Anaconda. Lo primero es crear un nuevo entorno:

```
conda create —name <nombre del entorno> python=3
```

Siempre que queramos volver a arrancar el entorno virtual (tras un reinicio, por ejemplo) tenemos que activarlo:

```
source activate <nombre del entorno>
```

Una vez tenemos el entorno creado, debemos configurarlo instalando las librerías

necesarias. Aquí hay dos opciones, las dos igual de válidas, para utilizar como backend de Keras: Theano y Tensorflow:

Si queremos utilizar Theano:

```
conda install theano
```

Si queremos utilizar Tensorflow[1]:

```
conda install tensorflow-gpu
```

Luego instalamos el resto de librerías necesarias:

```
conda install web_en_core_md spacy numpy keras
```

Ya lo tenemos todo configurado, ahora podemos comenzar a entrenar. Lo primero será generar los sentence embeddings de la siguiente manera:

```
python sentenceEmbedding_classification.py
```

Ésto nos genera varios archivos que se utilizarán en las siguientes etapas. Lo siguiente es entrenar el modelo:

```
python main_classification.py
```

El tiempo de entrenamiento depende del número de epochs, del tamaño del batch[2] y de nuestra tarjeta gráfica. Con un batch de 5, 200 epochs en una NVIDIA GTX 1080 el entrenamiento tarda aproximadamente 10 minutos. En Linux podemos ver el estado de la tarjeta gráfica mediante el comando **nvidia-smi**, en Windows existen varias herramientas, nosotros hemos utilizado **Graphics Engine** de Gigabyte.

Por último, para evaluar la calidad de las respuestas generadas ejecutamos el siguiente comando:

```
python evaluate_classification.py
```

Este programa imprimirá las preguntas del conjunto de datos de test junto con las respuestas generadas. Además nos da un cómputo total de respuestas acertadas y respuestas erróneas.

[1] También se encuentra disponible el paquete tensorflow que utiliza la CPU en lugar de la GPU

[2] ambos parámetros son configurables desde el código de main_classification.py, mediante las variables NUM_EPOCHS y BATCH_SIZE