TRABAJO FIN DE GRADO

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS

Análisis de opiniones

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

Autor
Nuria Rodríguez Barroso

Directores
Francisco Herrera Triguero
Eugenio Martínez Cámara

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Granada, septiembre de 2018

Agradecimientos

AGRADECIMIENTOS

Resumen

RESUMEN EN ESPAÑOL

Abstract

RESUMEN EN INGLES

Consentimientos

Índice general

Capítulo 1

Introducción

1.1 Contexto

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

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

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

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

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

1.2 Motivación

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

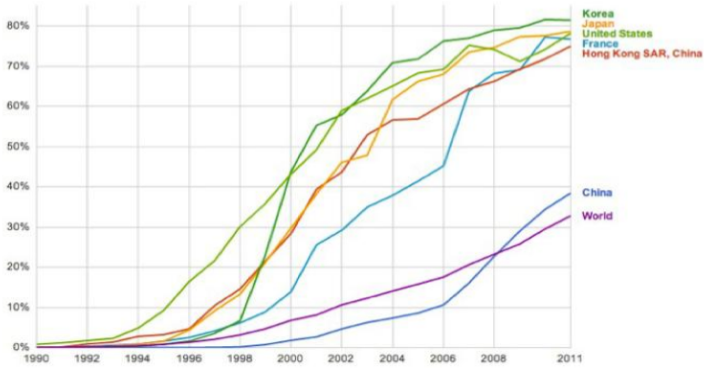


Figura 1.1: Gráfica del crecimiento del *Big Data* en algunos países y en el mundo.

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

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

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

El término de Análisis de Sentimientos es un término relativamente reciente. Aunque ya haya tenido éxito en algunos ámbitos, es un campo aún sin explotar pues presenta dificultades que aún no han sido solventadas. Entre estas dificultades se encuentran la subjetividad de las opiniones, las particularidades del lenguaje, los contextos, los sarcasmos ... entre una interminable lista.

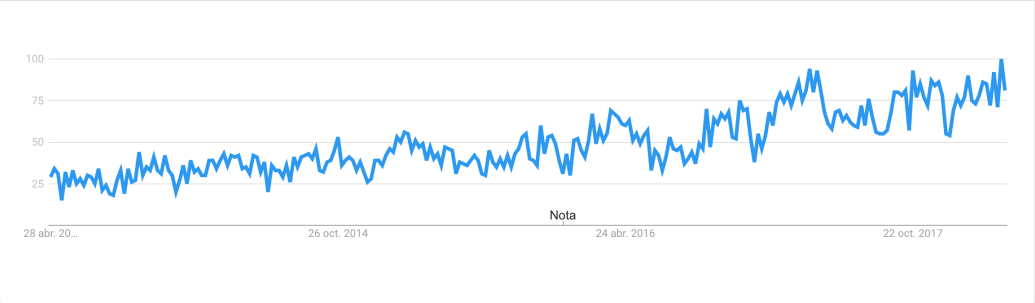


Figura 1.2: Gráfica que representa el interés del análisis de sentimientos en todo el mundo según Google Trends.

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

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

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

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

Si tuviéramos que establecer una polaridad a dichar frase, sería una tarea complicada incluso para una persona. En la frase se nombran tres componentes de un mismo ordenador y se da un opinión sobre cada una de ellas. Así, *buen procesador* tendría connotaciones positivas mientras que *la pantalla tiene poca resolución* y *el precio es muy elevado* negativas. La polaridad general del producto dependerá de la importancia que se le dé a cada una de las partes. Como no podemos saber a priori un orden de prioridad establecido, sería muy útil poder dar una polaridad a cada una de las componentes.

Ahora bien, para poder realizar este proceso de forma automatizada nos

encontraríamos con dos tareas: en primer lugar la identificación del aspecto y, en segundo lugar, la detección de la polaridad de esta. En este trabajo nos centraremos en la primera de estas tareas.

1.3 Objetivos

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

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

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

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

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

1.4 Requerimientos

Los requerimientos del proyecto que desarrollaremos se resumen en:

1. Clasificar/extraer entidades o aspectos mediante una arquitetura basada en CNN.

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

Capítulo 2

Análisis de Opiniones

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

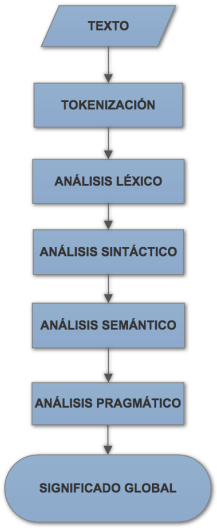
2.1 Procesamiento del Lenguaje Natural

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

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

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

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



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

2.2 Concepto de Análisis de opiniones

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

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

2.2.1 Concepto de Opinión

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

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

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

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

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

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

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

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

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

2.2.2 Niveles de clasificación

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

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

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

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

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

Capítulo 3

Fundamentos matemáticos

Concepto de derivada y definiciones alternativas.

Uso de la derivada en minimización de funciones.

matriz jacobiana

Gradiente para una función en Rn

Composición de funciones.

Grafos dirigidos.

ÁLGEBRA TENSORIAL – Optimización de algoritmos deeplearning (tensorflow)

PROBABILIDAD y TEORIA DE LA INFORMACION

- entropia cruzada – log-verosimilitud negativa

3.1 Probabilidad

- dist probabilidad

- estadistico – inferencia

Capítulo 4

Aprendizaje Automático

En este capítulo desarrollaremos los principios básicos de aprendizaje automático para introducir el *Deep Learning* en capítulos posteriores. Motivaremos su uso con algunos ejemplos y nos adentraremos en el problema de clasificación pues es el que desarrollaremos durante el resto de la memoria. erosi Finalmente hablaremos del gradiente descendente, algoritmo de optimización en el cual se inspirará el Deep Learning.

4.1 Concepto de Aprendizaje Automático

Un algoritmo de **Aprendizaje Automático** es un algoritmo que es capaz de aprender información a partir de ciertos datos. Pero, ¿qué entendemos por aprender? Según [Mitchell, 1997], "Un programa de ordenador se dice que aprende de una experiencia E con respecto a un conjunto de tareas T y medida de rendimiento P, si su rendimiento como tarea en T, medido con P, mejora tras conocer la experiencia E". Como podemos imaginar, estas tareas y experiencias son muy diversas, de donde surgen gran variedad de técnicas de aprendizaje con estas características.

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

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

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

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

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

4.1.1 Tipos de aprendizaje automático.

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

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

en la agrupación de los datos en diferentes subconjutnos que contengan características similares.

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

Prácticamente todo el Deep Learning está motivado por el **Gradiente Descendente Estocástico**. Este algoritmo es una extensión del Gradiente Descendente. Por tanto, a continuación haremos una breve introducción a estos dos conceptos.

4.2 Problema de clasificación

Encontrar referencia formal donde pueda yo explicar esto

4.3 Optimización basada en Gradiente Descendente

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

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

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

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

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

$$x' = x - \epsilon \nabla_x f(x)$$

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

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

Ejemplos?

4.3.1 Gradiente Descendente Estocástico

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

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

De esta forma, la estimación del gradiente sería

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

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

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

donde ϵ es la tasa de aprendizaje.

El Deep Learning se ve motivado porque el Aprendizaje Automático no había conseguido resolver los problemas centrales de la Inteligencia Artificial, como por ejemplo el reconocimiento de imágenes o procesamiento natural del lenguaje.

Capítulo 5

Deep Learning

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

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

5.1 Redes Neuronales Prealimentadas

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

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

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

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

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

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

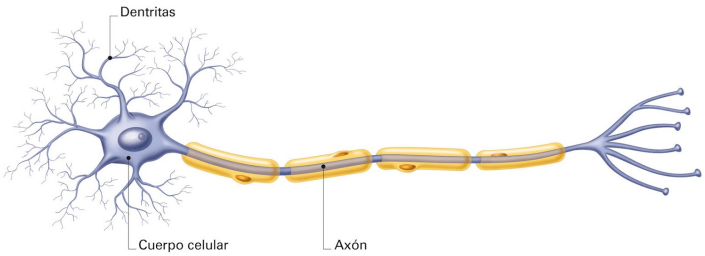


Figura 5.1: Representación de una neurona biológica.

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

La neurona artificial se define como en la Figura 5.2, donde identificamos una neurona biológica con una función $f : \mathbb{R}^n \to \mathbb{R}$ donde n es el número de unidades en la capa anterior (entradas).

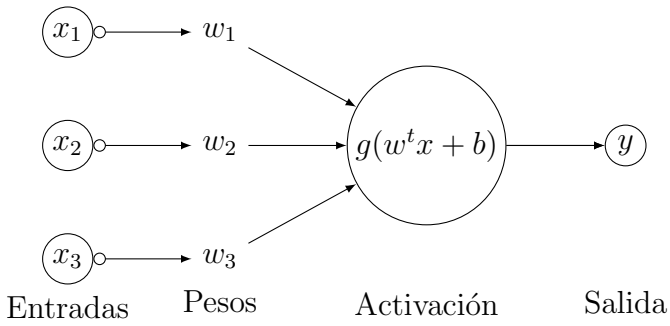


Figura 5.2: Representación de una neurona artificial de 3 entradas con función de activación g.

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

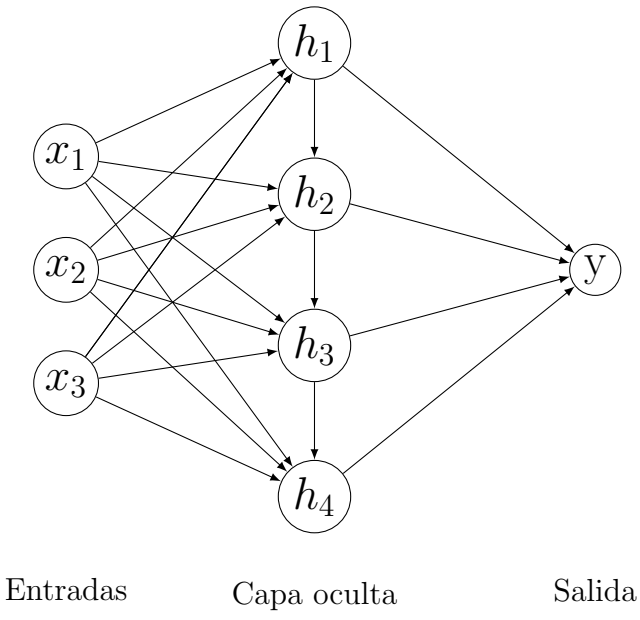


Figura 5.3: Representación de una neurona artificial de 3 entradas con función de activación g.

Observamos en la Figura 5.3 un ejemplo simplificado de una red neuronal con tres capas: capa de entrada, una capa oculta y capa de salida. La capa

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

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

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

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

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

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

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

A continuación estudiaremos algunas de las funciones comentadas.

5.1.1 Función de coste

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

Aunque en algunos casos podemos definir la función de coste como un estadístico de y condicionado a x. En la mayoría de los diseños, el modelo paramétrico definido define una distribución de probabilidad $p(y|x;\theta)$ y podemos aplicar el principio de máxima verosimilitud comentado en la Sección 3.1. Esto significa que usamos la entropía cruzada entre los datos de entrenamiento y las predicciones del modelo como función de coste:

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

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

5.1.2 Unidades de salida

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

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

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

Unidades lineales

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

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

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

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

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

Unidades sigmoidales

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

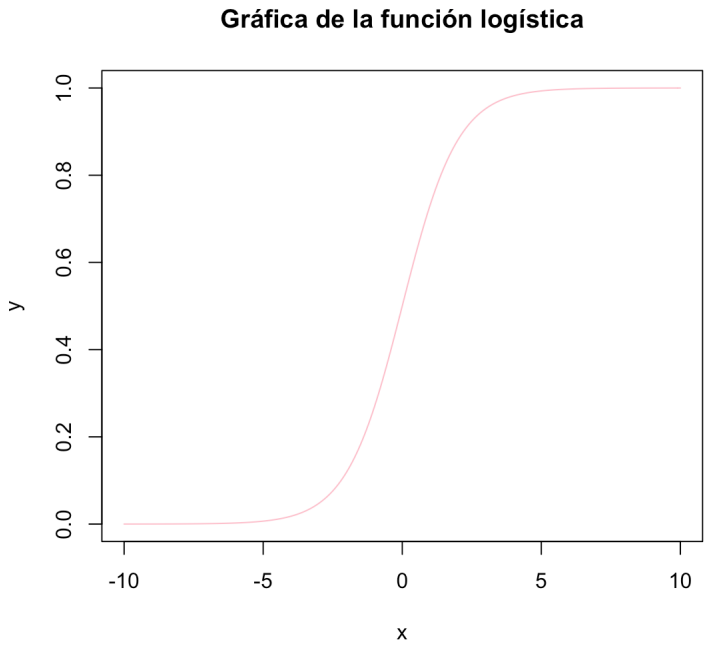


Figura 5.4: Gráfica de la función logística. Toma valores en (0,1)

Unidades con activación softmax

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

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

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

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

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

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

Pasando a exponencial y normalizando obtenemos la función *softmax*

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

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

5.1.3 Unidades ocultas

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

Unidades lineales rectificadas (ReLu) y generalizaciones

Las unidades lineales rectificadas usan como función de activación

$$g(z) = \text{máx}\{0, z\}$$

Estas unidades son fácil de optimizar dado que son muy similares a las unidades lineales. La única diferencia radica en que la unidad lineal rectificada toma el valor 0 en la mitad del su dominio. Observamos la gráfica en la Figura 5.5.

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

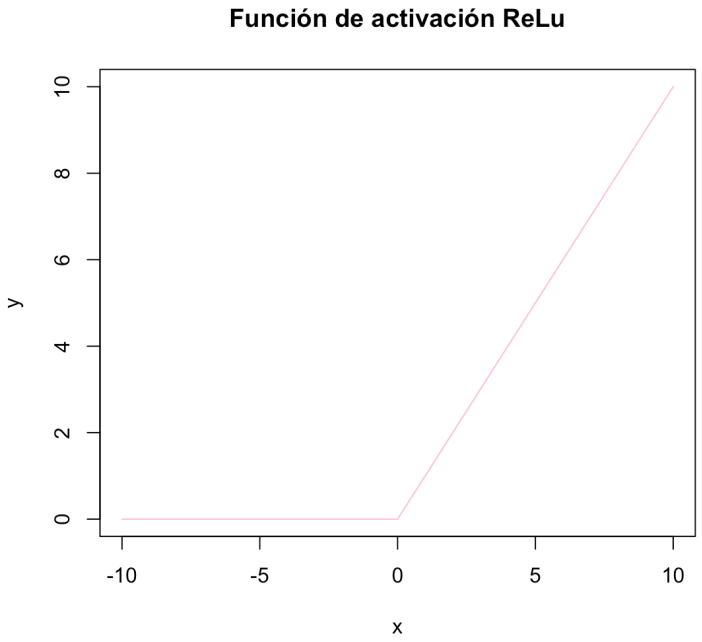


Figura 5.5: Gráfico de la función de activación de las unidades lineales rectificadas.

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

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

Sigmoidal y tangente hiperbólica

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

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

cuya gráfica podemos observar en la Figura 5.4 o la tangente hiperbólica como función de activación

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

cuya gráfica es

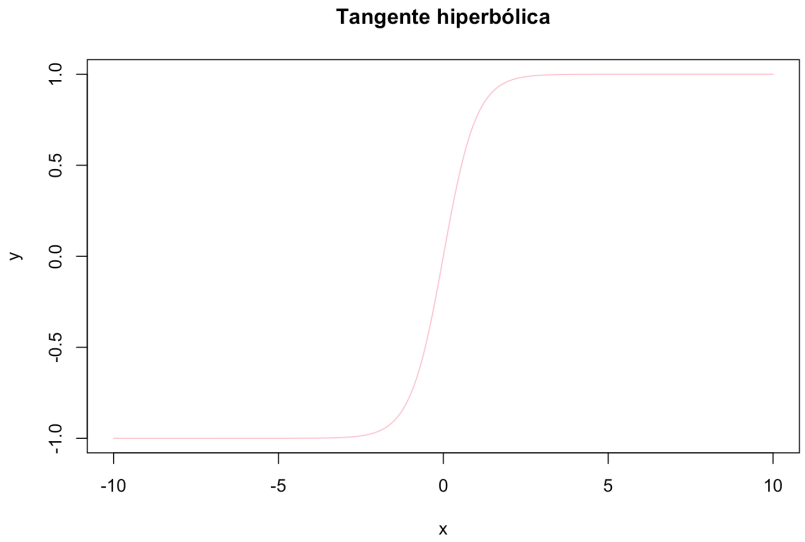


Figura 5.6: Gráfica de la tangente hiperbólica.

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

Capítulo 6

Redes Neuronales Convolucionales

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

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

6.1 La operación de Convolución

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

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

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

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

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

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

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

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

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

$$S(i,j) = (I*K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) = \sum_m \sum_n K(m,n)I(i-m, j-n)$$

6.1.1 Ejemplo de convolución

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

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

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

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

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

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

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

6.2 *Pooling*

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

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

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

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

6.3 Adaptación a procesamiento de datos secuencial

Capítulo 7

Modelado de secuencias: Redes Neuronales Recurrentes

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

7.1 Motivación

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

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

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

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

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

Al igual que en capítulo 6 se definía la estructura del modelo de las redes neuronales convolucionales como un grafo acíclico, la recurrencia de este tipo de redes ampliará el concepto a grafos cíclicos. Estos ciclos representarán la influencia del valor actual de la variable en su propio valor en el siguiente paso del aprendizaje.

7.2 Despliegue de Grafos Computacionales

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

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

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

$$s^{(t)} = f(s^{(t-1)}; \theta) \tag{7.1}$$

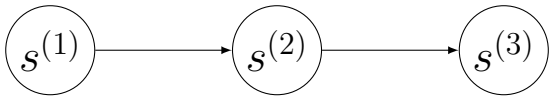
donde $s^{(t)}$ es el estado del sistema.

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

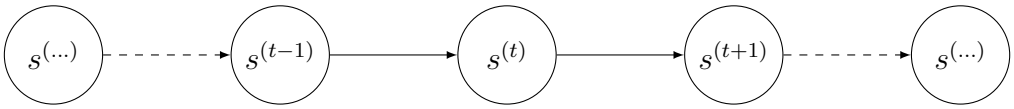
Si consideramos un número de pasos finito τ, el grafo puede ser desplegado aplicando la definición $\tau - 1$ veces. Por ejemplo, para $\tau = 3$ tenemos:

$$s^{(3)} = f(s^{(2)}; \theta) = f(f(s^{(1)}; \theta); \theta) \tag{7.2}$$

Desplegando sucesivamente la fórmula con la definición de s hemos obtenido una expresión que no involucra recurrencia y que puede ser expresada con un gráfico acíclico. De esta forma el grafo para la fórmula 7.2 sería:



Y de forma genérica, para la fórmula 7.1 tendríamos el siguiente grafo



La mayoría de las redes neuronales recurrentes utilizan la fórmula 7.3 para definir el valor de salida de las capas ocultas.

$$h^{(t)} = f(h^{(t-1)}, x^{(t)}; \theta) \tag{7.3}$$

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

La ecuación 7.3 se puede representar de dos formas diferentes:

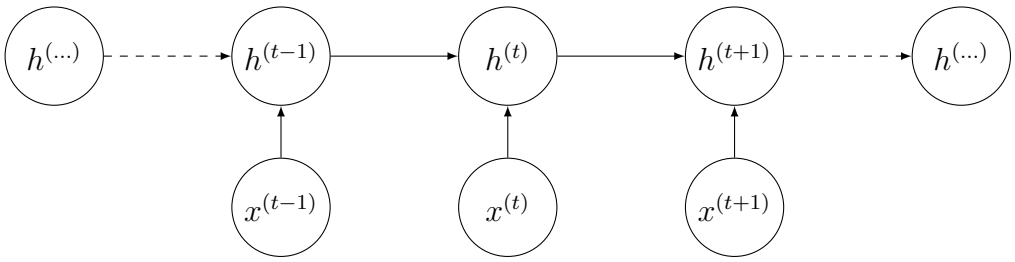


Figura 7.1: Representación extendida de las operaciones de las capas ocultas en una red neuronal recursiva.

1. Considerando un nodo por cada uno de los estados por los que pasa cada una de las variables del modelo como en la Figura 7.1.

2. Considerando una forma resumida, donde cada nodo representa las componentes que existirían en una implementación física del modelo como en la Figura 7.2.

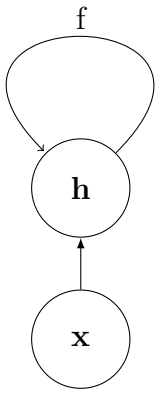


Figura 7.2: Representación esquematizada de las operaciones de las capas ocultas en una red neuronal recursiva.

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

7.3 Redes Neuronales Recurrentes

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

Podemos observar un esquema de red recurrente en la Figura 7.3.

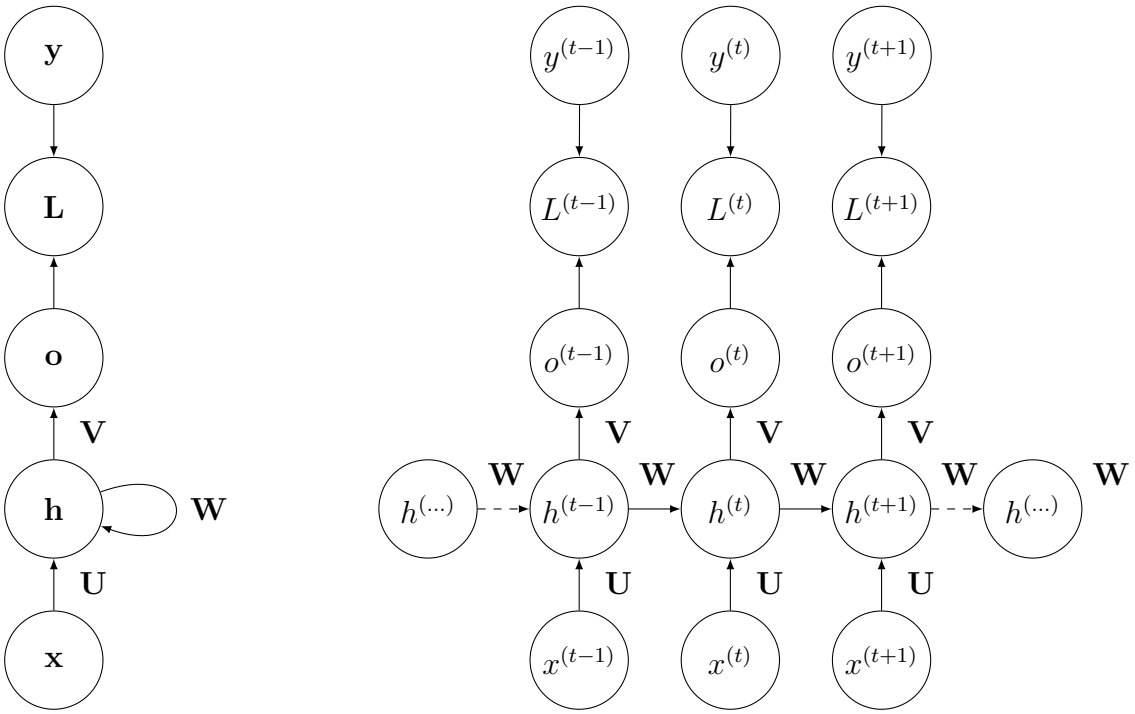


Figura 7.3: Representaciones de una red neuronal recurrente genérica.

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

Una vez establecida la arquitectura de la red, nos centramos en desarrollar las ecuaciones propagación hacia delante (*forward propagation*). En este ca-

so supondremos que la función de activación de las capas ocultas será una tangente hiperbólica (*tanh*) y que la salida es discreta. Una forma común de representar variables discretas es considerar la salida **o** como una probabilidad logarítmica no normalizada para cada posible valor de la variable discreta. Entonces, podemos aplicar una operación *softmax* a dichas probabilidades como se explica en la sección 5.1.2.

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

$$\mathbf{a}^{(t)} = \mathbf{b} + \mathbf{W}\ \mathbf{h}^{(t-1)} +\ \mathbf{U}\mathbf{x}^{(t)} \tag{7.4}$$

$$\mathbf{h}^{(t)} = \tanh(\mathbf{a}^{(t)}) \tag{7.5}$$

$$\mathbf{o}^{(t)} = \mathbf{c} + \mathbf{V}\mathbf{h}^{(t)} \tag{7.6}$$

$$\hat{\mathbf{y}}^{(t)} = softmax(\mathbf{o}^{(t)}) \tag{7.7}$$

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

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

$$\mathcal{L}(\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(\tau)}\}, \{\mathbf{y}^{(1)}, ..., \mathbf{y}^{(\tau)}\}) = \sum_t \mathcal{L}^{(t)}) = -\sum_t \log p(y^{(t)} | \mathbf{x}^{(1)}, ..., \mathbf{x}^{(t)}\}) \tag{7.8}$$

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

cada paso de tiempo tiene que ser ejecutado después de haber ejecutado el anterior. Los cálculos realizados en la propagación hacia delante se almacenan utilizándose en la propagación hacia atrás, así conseguimos una coste de memoria de $\mathcal{O}(\tau)$ nuevamente.

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

7.3.1 Cálculo del Gradiente

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

Para entender mejor el comportamiento del método $BPTT$, lo aplicaremos sobre las ecuaciones 7.4 a la 7.7:

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

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

$$\frac{\partial L}{\partial L^{(t)}} = 1 \tag{7.9}$$

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

$$(\nabla_{\mathbf{o}^{(t)}} L)_i = \frac{\partial L}{\partial o_i(t)} = \frac{\partial L}{\partial L^{(t)}} \frac{\partial L^{(t)}}{\partial o_i(t)} = \hat{y}_i^{(t)} - \mathbf{1}_{i,y^{(t)}} \tag{7.10}$$

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

$$\nabla_{\mathbf{h}^{(\tau)}} L = \mathbf{V}^T \nabla_{\mathbf{o}^{(\tau)}} L \tag{7.11}$$

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

$$\nabla_{\mathbf{h}^{(t)}} L = \left(\frac{\partial \mathbf{h}^{(t+1)}}{\partial \mathbf{h}^{(t)}}\right)^T (\nabla_{\mathbf{h}^{(t+1)}} L) + \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{h}^{(t)}}\right)^T (\nabla_{\mathbf{o}^{(t)}} L) = \tag{7.12}$$

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

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

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

$$\nabla_{\mathbf{c}} L = \sum_t \left(\frac{\partial \mathbf{o}^{(t)}}{\partial \mathbf{c}}\right)^T \nabla_{\mathbf{o}^{(t)}} L = \sum_t \nabla_{\mathbf{o}^{(t)}} L \tag{7.13}$$

$$\nabla_{\mathbf{b}} L = \sum_t \left(\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{b}^{(t)}}\right)^T \nabla_{\mathbf{h}^{(t)}} L = \sum_t diag\left(1 - \left(\mathbf{h}^{(t)}\right)^2\right) \nabla_{\mathbf{h}^{(t)}} L \tag{7.14}$$

$$\nabla_{\mathbf{v}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_o^{(t)}}\right) \nabla_{\mathbf{v}} o_i^{(t)} = \sum_t (\nabla_{\mathbf{o}^{(t)}} L) \mathbf{h}^{(t)^T} \tag{7.15}$$

$$\nabla_{\mathbf{W}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{\mathbf{W}^{(t)}} h_i^{(t)} = \sum_t diag\left(1 - \left(\mathbf{h}^{(t)}\right)^2\right) (\nabla_{\mathbf{h}^{(t)}} L) \mathbf{h}^{(t-1)^T} \tag{7.16}$$

$$\nabla_{\mathbf{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}} \right) \nabla_{\mathbf{U}^{(t)}} h_i^{(t)} = \sum_t diag\left(1 - \left(h^(t)\right)^2\right) (\nabla_{\mathbf{h}^t} L) \mathbf{x}^{(t)^T} \tag{7.17}$$

7.3.2 El problema de las dependencias a largo plazo

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

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

$$\mathbf{h}^{(t)} = \mathbf{W}^T \mathbf{h}^{(t-1)} \tag{7.18}$$

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

$$\mathbf{h}^{(t)} = (\mathbf{W}^t)^T \mathbf{h}^{(0)}, \tag{7.19}$$

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

$$\mathbf{W} = \mathbf{Q} \Lambda \mathbf{Q}^T \tag{7.20}$$

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

$$\mathbf{h}^{(t)} = \mathbf{Q}^T \Lambda^t \mathbf{Q} \mathbf{h}^{(0)} \tag{7.21}$$

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

7.3.3 Redes neuronales recurrentes con memoria a corto-largo plazo

Las redes neuronales recurrentes con memoria a corto-largo plazo o *long short-term memory (LSTM)* tratan de solucionar el problema de las dependencias a largo plazo 7.3.2.

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

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

En la Figura 7.4 se esquematiza el diagrama de bloque que representa una celda de una *LSTM*. Las celdas se conectan entre ellas recurrentemente, reemplazando las unidades ocultas de las redes neuronales comunes. Estas celdas tienen una recurrencia interna (*self-loop*) además de la recurrencia interna entre celdas. Cada celda tiene las mismas entradas y salidas que una red recurrente normal, pero tiene más parámetros y sistemas de puertas que controlan el flujo de información.

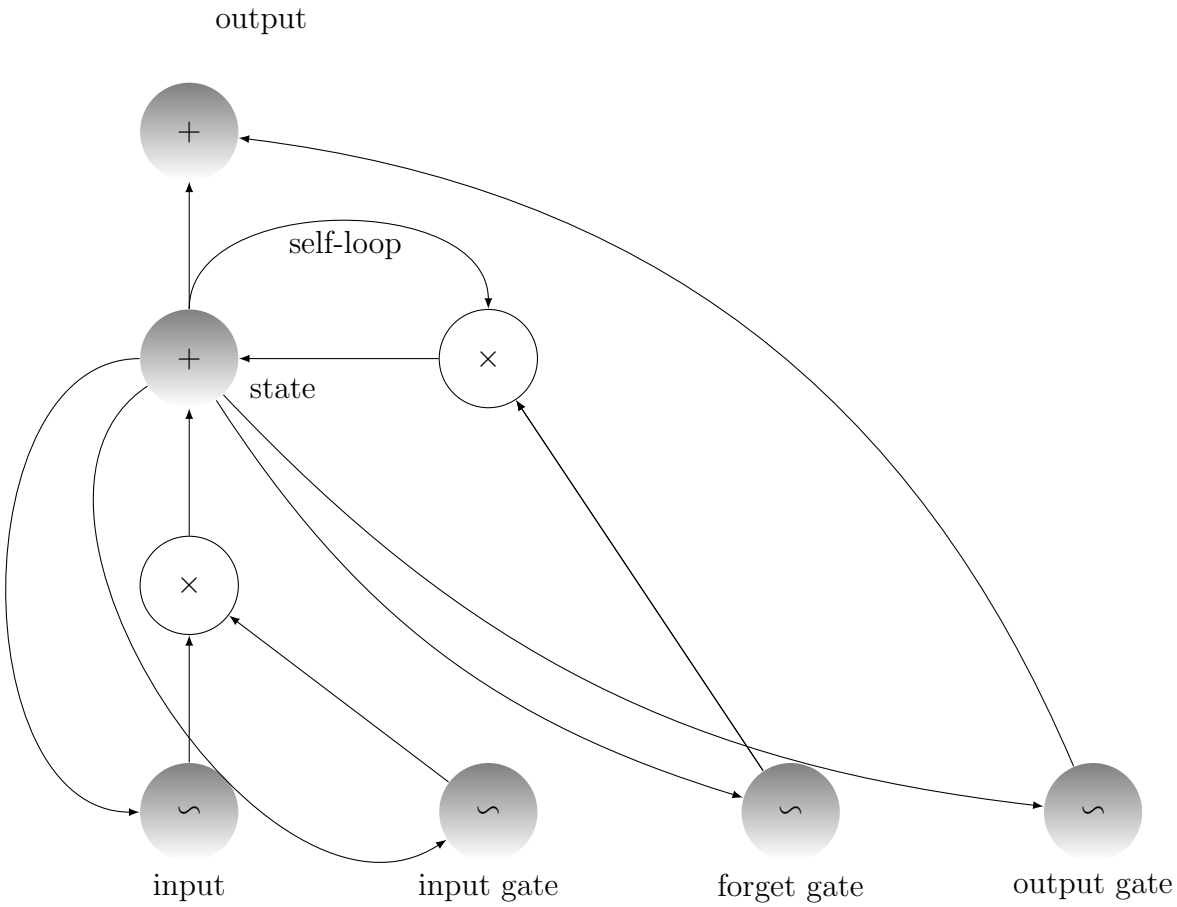


Figura 7.4: Representaciones de una red neuronal recurrente genérica.

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

$$f_i^{(t)} = \sigma\left(b_i^f + \sum_j U_{i,j}^f x_j^{(t)} + \sum_j W_{i,j}^f h_j^{(t-1)}\right), \tag{7.22}$$

donde $\mathbf{x}^{(t)}$ es el actual vector de entrada y $\mathbf{h}^{(t)}$ es el vector de la actual capa

oculta, que contiene las salidas de todas las celdas previas y $\mathbf{b}^{(t)}$, $\mathbf{U}^{(t)}$ y $\mathbf{W}^{(t)}$ son los sesgos, pesos de entrada y pesos recurrentes de las unidades de olvido respectivamente.

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

$$s_i^{(t)} = f_i^{(t)} s_i^{(t-1)} + g_i^{(t)} \sigma \left(b_i + \sum_j U_{i,j} x_j^{(t)} + \sum_j W_{i,j} h_j^{(t-1)} \right) \tag{7.23}$$

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

$$g_i^{(t)} = \sigma \left(b_i^g + \sum_j U_{i,j}^g x_j^{(t)} + \sum_j W_{i,j}^g h_j^{(t-1)} \right) \tag{7.24}$$

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

$$h_i^{(t)} = \tanh \left(s_i^{(t)} \right) q_i^{(t)}, \tag{7.25}$$

$$q_i^{(t)} = \sigma \left(b_i^o + \sum_j U_{i,j}^o x_j^{(t)} + \sum_j W_{i,j}^o h_j^{(t-1)} \right) \tag{7.26}$$

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

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

Capítulo 8

Primera arquitectura

Capítulo 9

Segunda arquitectura

Bibliografía

[Mitchell, 1997] *Machine Learning.* McGraw-Hill, New York.

[Goodfellow, 2016] *Deep Learning.* Gooldfellow, Ian. |Bengio, Yosuha. |Courville, Aaron.