



UNIVERSIDAD
DE GRANADA

Escuela Técnica Superior de Ingenierías Informática y de
Telecomunicación
Facultad de Ciencias

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y
MATEMÁTICAS

TRABAJO DE FIN DE GRADO

Optimización de redes neuronales

Presentado por:
Blanca Cano Camarero

Tutores:
Juan Julián Merelo Guervós
Arquitectura y tecnología de computadores

Francisco Javier Meri de la Maza
Análisis matemático

Curso académico 2021-2022

Optimización de redes neuronales

Blanca Cano Camarero

Blanca Cano Camarero *Optimización de redes neuronales.*
Trabajo de fin de Grado. Curso académico 2021-2022.

**Responsables de
tutorización**

Juan Julián Merelo Guervós
Arquitectura y tecnología de computadores

Francisco Javier Meri de la Maza
Análisis matemático

Doble Grado en Ingeniería
Informática y Matemáticas

Escuela Técnica Superior
de Ingenierías Informática
y de Telecomunicación
Facultad de Ciencias

Universidad de Granada

Índice general

1 Metodología	13
1.1 <i>Personas definidas</i>	13
1.2 Historias de usuario	14
1.3 Hitos	16
1.4 Registro de trabajo	17
1.5 Resumen de la metodología	17
1.6 Herramientas utilizadas	18
1.6.1 GitHub	18
1.6.2 Lenguaje de programación Julia	18
I Teoría subyacente	19
2 Introducción a las redes neuronales	23
2.1 Concepto de aprendizaje	24
2.1.1 Componentes del aprendizaje	24
2.1.2 Tipos de aprendizaje	25
3 Teoría de la aproximación	27
3.1 Polinomios de Bernstein	27
3.2 Teorema de Stone-Weierstrass	32
3.3 Conclusiones teoría de la aproximación	34
4 Las redes neuronales son aproximadores universales	39
4.1 Definición de las redes neuronales <i>Feedforward Networks</i> de una capa oculta	39
4.2 Las redes neuronales son aproximadores universales	41
4.3 Definiciones primeras	42
4.3.1 Reflexión sobre el tipo de funciones que se pueden aproximar	44
4.4 Primeros resultados	45
4.4.1 Observaciones y reflexiones sobre el teorema de convergencia real en compactos	46
4.5 Generalización a espacios L_p	59
4.5.1 Reflexión sobre el número de neuronas	65
4.6 Generalización para <i>multi-output neuronal networks</i>	65
4.7 Consideración sobre la capacidad de cálculo	68
5 Construcción técnica de las redes neuronales de una sola capa	71
5.1 Componentes de una red neuronal de una capa oculta	71

5.1.1	Consideraciones sobre la irrelevancia del sesgo	72
5.1.2	<i>Modus operandi</i> ante problemas que requieran un dominio de salida discreto	74
5.1.3	Qué función de activación seleccionar	76
5.2	Construcción explícita y evaluación de una red neuronal	76
5.2.1	Implementación de una red neuronal y evaluación	78
5.3	Aprendizaje	80
5.3.1	Método de gradiente descendente y <i>backpropagation</i>	80
5.3.2	Motivación para almacenar cálculos parciales	84
5.3.3	Algoritmos de actualización de pesos de una neurona	84
5.3.4	Otras alternativas al alternativo de gradiente descendente	87
5.4	Inicialización de los pesos de una red neuronal	88
5.4.1	Estado del arte relacionado	88
5.4.2	Descripción del método propuesto	88
II	Estudio experimental y exploración hipótesis planteadas	93
6	Funciones de activación	95
6.1	Caracterización de las funciones de activación	95
6.1.1	Implementación de las funciones de activación en la biblioteca de redes neuronales	98
6.1.2	Coste computacional funciones activación	98
7	Hipótesis de optimización	103
7.1	Inicialización de la pesos red neuronal	103
7.2	Algoritmo genéticos	103
	Bibliografía	105

Índice de figuras

3.1	Ejemplo de aproximación de la función $f(x)$ a partir de los polinomios de Lagrange.	35
3.2	Cómo actúa en la aproximación una función de activación	36
3.3	Cómo afecta la forma de la función de activación	37
4.1	<i>Grafo</i> de una red neuronal de una capa oculta	39
4.2	Ejemplos de funciones de activación	42
4.3	Función <i>cosine squasher</i>	53
4.4	Comparativa $H_{[-\pi,0]}$ con la función coseno real.	54
4.5	Función $H_{[-\pi,\pi]}$	55
4.6	Ejemplos de funciones $H_{[-M,M]}$	55

4.7	Ejemplo de red neuronal de una capa oculta con n nodos, de dimensión de entrada d y salida s	66
5.1	<i>Grafo</i> de una red neuronal de una capa oculta	71
5.2	Ejemplo de evaluación de una red neuronal de una capa con entrada y salida de tamaño dos y dos neuronas en la capa oculta	78

Índice de tablas




5.1	Coste computacional de la implementación propuesta en [AMMIL12]	77
5.2	Coste computacional de nuestra implementación de red neuronal de una capa	77
5.3	Comparativas de coste computacional entre la implementación usual de una red neuronal y la nuestra	77
5.4	Coste computacional de aplicar directamente el algoritmo de gradiente descendente para actualizar $h \in \mathcal{H}_n(\mathbb{R}^d, \mathbb{R}^s)$	83
5.5	Veces que se calcula una misma expresión para el cálculo de gradiente descendente fijado un i y coste en memoria en unidades si se almacenara el cálculo de todos esos parámetros.	83
5.6	Valor mínimo del parámetro M en algoritmo de inicialización de redes neuronales según la función de activación seleccionada.	91
5.7	Valor mínimo del parámetro M en algoritmo de inicialización de redes neuronales según la función de activación seleccionada (con más resultados).	92
6.1	Compendio de funciones de activación	96
6.2	Agrupaciones de funciones de activación con forma similar	98
6.3	Resultados 1 de 3: Rechazos con un 95 % de confianza en el test Wilcoxon.	99
6.4	Resultados 2 de 3: Rechazos con un 95 % de confianza en el test Wilcoxon.	100
6.5	Resultados 3 de 3: Rechazos con un 95 % de confianza en el test Wilcoxon.	100
6.6	Tiempo de ejecución en segundos	101

Índice de algoritmos

1	Cálculo de la función θ como clasificador 1–NN	75
2	Estructura de una red neuronal	79
3	Evaluación de una red neuronal, <i>Forward propagation</i>	79
4	Algoritmo gradiente descendente conocidas las derivadas parciales.	85
5	Algoritmo cálculo del gradiente $\nabla E(h)$	86
6	Inicialización de pesos de una red neuronal	90

Comentario previo

Se pretende con este documento presentar un trabajo de fin del doble grado de Ingeniería Informática y Matemáticas y que cualquier miembro del tribunal; independientemente de la vertiente a la que pertenezca sea capaz de comprenderlo en su totalidad. Para ello se ha acompañado la exposición con notas en los márgenes aclaratorias, que siguen el siguiente código de color y icono ¹:

-  **Color 1:** Comentarios para aclarar conceptos matemáticos o informáticos y ofertar la idea intuitiva que se esconde, donde no se presuponen conocimientos avanzados en la materia.
-  **Color 2:** Comentarios para una reflexión más profunda o que indique nuevas áreas que explorar.
-  **Color 3:** Concepto clave y destacable que tendrá un papel fundamental a posteriori.

Además a lo largo del trabajo se han realizado aportaciones propias y resultados novedosos, estos aparecerán destacados de la siguiente forma:

Aportación original

Este recuadro representa cómo se indicará de ahora en adelante alguna aportación original a la materia.

¹Como parte de la metodología los colores han sido seleccionados de acorde a una paleta de colores inclusiva. Para ello se ha seleccionado una paleta de la web Palett.es, visitada por última vez el 13 de mayo del 2022 y con dirección <https://palett.es/6a94a8-013e3b-7eb645-31d331-26f27d>.

1. Metodología

La planificación y organización es un componente vital a la hora del desarrollo de software y la ciencia de datos, por lo que no es de extrañar que sus beneficios sean extrapolables a otras áreas de la ciencia; hemos aplicado por ende la metodología expuesta en el artículo [Gue21], realizado el trabajo desarrollando con una mentalidad ágil, de acorde a las siguientes hipótesis:

1. Se premia la **reproducibilidad**, *la ciencia no puede ser ciencia si no es reproducible*. Es por ello que se publica todo el material usado.
2. **Comprobaciones** en todo los niveles, *el código que no se ha probado no funciona*. Explicamos unos requisitos y criterios claros de validación.
3. Conocimiento libre, tanto la memoria como el código se encuentra publicado y con licencia libre en [nuestro repositorio de GitHub](#) ¹.
4. **Colaboración frente jerarquía de mando vertical**, los tutores en este caso los *interesados en el producto* indican qué les gustaría que tuviera en vez de un qué y cómo estricto. Consiguiendo con ello una mayor autonomía y bienestar de trabajo.

Concretamente la metodología seguida basada en [Gue] ha sido la siguiente:

El objetivo es *optimizar el proceso de diseño, entrenamiento y explotación* para reducir su coste de cómputo y espacio en memoria en base de un fundamento matemático sólido y sistemático que provenga de la noción más elemental de red neuronal y no meras heurísticas experimentales.

Puesto que los problemas surgen de las necesidades humanas, se han definido unas metas claras a través de la *metodología de personas* (ver sección 1.1) [D] y las respectivas historias de usuario derivadas (ver sección 1.2).

1.1. Personas definidas

Ante la pregunta de quiénes serán los beneficiados e involucrados por este proyecto y que trataremos de satisfacer han surgido las siguientes *personas ficticias*:

- **Rosa Camarero**, desea implementar un modelo de red neuronal que mejore a los actuales en algún aspecto, ya sea porque reduzca su coste computacional o espacio en memoria.
- **María López**, catedrática en matemáticas aplicadas y miembro del tribunal . Su conocimiento de informática es moderado. Valora una memoria formal y rigurosa. Se encarga de evaluar trabajos fin de grado, en particular éste mismo

¹Puede visitarlo en <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales>.

© El objetivo de la metodología seleccionada.

Buscamos organizarnos de una manera eficiente con el fin de optimizar el proceso de producción y conducir a un resultado exitoso. Para ello nos basaremos en:

1. ¿Qué personas se beneficiarán y están involucradas en el proyecto? (**Personas definidas 1.1**)

2. ¿Qué caracteriza a esas personas y cuáles son sus necesidades? (**Historias de usuario 1.2**)

3. ¿Qué acciones satisfarán tales requisitos?(**Hitos 1.3**)

1. Metodología

- **Pedro Castillo**, Doctor experto en *deep learning* y miembro del tribunal. Valora la claridad, las experimentaciones sensatas y fundamentadas así cómo la posibilidad de aplicaciones reales. Se encarga de evaluar trabajos fin de grado, en particular éste mismo.
- **Javier Merí**, profesor del departamento de análisis matemático y contutor del proyecto (junto con JJ). Busca de manera formal rigurosa entender y optimizar las redes neuronales usando herramientas analíticas. Estará presente durante todo el desarrollo.
- **JJ Merelo**, profesor del departamento de arquitectura de computadores y contutor del proyecto (junto con Javier). Busca resolver problemas relacionados con la optimización bajo un punto de vista más práctico y experimental, salvaguardando un trabajo bien organizado y metódico. Estará presente durante todo el desarrollo.
- **Blanca Cano** alumna de informáticas y matemáticas, tiene muchas ganas de aprender sobre redes neuronales, su conocimiento es moderado, espera presentar el trabajo en la convocatoria ordinaria.

1.2. Historias de usuario

A partir de los usuarios se han definido las historias de usuario [Coh9], que han quedado registradas como *issues* etiquetadas con *user story* y cabecera de formato [HUxx] título de la historia de usuario [nuestro repositorio de GitHub](#), donde HUxx representa la historia de usuario número xx.

Las historias de usuario definidas son las siguientes:

[HU01] Optimización redes neuronales.

Como *Rosa* me gustaría que se ejecutaran las redes neuronales en un hardware más barato o que consuma menos en un tiempo razonable. Puede encontrar su declaración en las historias de usuario definidas en [nuestro repositorio de GitHub](#) ².

[HU02] Metodología

Como *JJ* me gustaría que el proyecto se realizara bajo un desarrollo ágil, el cuál recoge sus principios en el siguiente manifiesto [Ken] y que es una filosofía, no una metodología [Rud] [Bos]. Puede encontrar su declaración en las historias de usuario definidas en [nuestro repositorio de GitHub](#) ³.

[HU03] Estudio de las redes neuronales

Como *Javier* me gustaría que la memoria fuera un estudio de calidad de las redes neuronales, respondiendo de manera auto-contenida, clara, rigurosa y precisa a preguntas como:

- ¿Qué son las redes neuronales?
- ¿Cómo se construyen?

²Visite el *issue* 48 <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales/issues/48>.

³Visite el *issue* 65 <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales/issues/65>.

- ¿Para qué sirven?
- ¿Por qué funcionan?

Puede encontrar su declaración en las historias de usuario definidas en [nuestro repositorio de GitHub](#) ⁴.

[HU04] Indagar y experimentar sobre redes neuronales

Como *Javier y JJ* gustaría plantear nuevas hipótesis para conocer en mayor profundidad, aclarar limitaciones, particularizar o abstraer en cualquier aspecto de las redes neuronales actuales.

Puede encontrar su declaración en las historias de usuario definidas en [nuestro repositorio de GitHub](#) ⁵.

[HU05] Requisitos tribunal

Como *Mayte y Pablo* queremos corregir un trabajo fin de grado bien hecho. Bajo el contexto de evaluación la memoria debe de cumplir una serie de requisitos imprescindibles:

1. Declaración explícita firmada en la que se asume la originalidad del trabajo, entendida en el sentido de que no ha utilizado fuentes sin citarlas debidamente. Esta declaración se puede descargar en la web del Grado (matemáticas).
2. Un índice detallado de capítulos y secciones.
3. Un resumen amplio en inglés del trabajo realizado (se recomienda entre 800 y 1500 palabras).
4. Una introducción en la que se describan claramente los objetivos previstos inicialmente en la propuesta de TFG, indicando si han sido o no alcanzados, los antecedentes importantes para el desarrollo, los resultados obtenidos, en su caso, y las principales fuentes consultadas.
5. Una descripción de la metodología.
6. Un estado del arte y diseño.
7. Estimación de los costes.
8. Análisis de las prestaciones.
9. Una bibliografía final que incluya todas las referencias utilizadas.

Puede encontrar su declaración en las historias de usuario definidas en [nuestro repositorio de GitHub](#) ⁶.

Estas historias de usuario dan lugar a *hitos*.

⁴Visite el *issue* 50 <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales/issues/50>.

⁵Visite el *issue* 51 <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales/issues/51>.

⁶Visite el *issue* 49 <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales/issues/49>.

1.3. Hitos

Los hitos describen entregables, productos mínimos viables y se basan en las historias de usuarios, puede encontrar nuestras declaraciones en nuestro repositorio ⁷. A éstas además le hemos asociado una fecha de entrega.

Hito 0: Explicitación escrita de la metodología

Consiste en la redacción de la descripción de la metodología a seguir resultando en capítulos concretos de la memoria y *issues* en la plataforma de trabajo, GitHub. La documentación presentada deberá de incluir:

- Explicitación de la metodología.
- Definición y descripción de personas.
- Redacción de las historias de usuario.
- Redacción de los hitos.

Será validada cuando los tutores aprueben la organización y formulación de la misma.

Hito 1: Búsqueda de resultados teóricos para optimizar las redes neuronales

Se busca con este hito establecer las bases teóricas para poder optimizar las redes neuronales, se dará por concluido cuando se haya conseguido algún resultado teórico con el que:

- Acotar el tamaño o la precisión de los pesos u otros parámetros de la red neuronal.
- Alguna forma de acelerar la convergencia, si es que se va a optar por un método de descenso de gradiente.
- En general, cualquier hipótesis que se pueda relajar o cambiar en el camino de reducir el espacio de búsqueda de posibles redes neuronales o de los pesos de las mismas.

El estudio progresivo se verá reflejado en varios capítulos y secciones de la memoria. Con secciones como: *Definición y construcción de las redes neuronales*. Este hito consiste en la redacción de una serie de capítulos y secciones que estudien:

Se aceptarán las secciones una vez hayan sido supervisadas y validadas por los tutores.

Hito 2: Evaluación experimental de las hipótesis de optimización formuladas

Deberá de validarse y cuantificar experimentalmente las propuestas de optimización del hito anterior.

Para ello se deberá de:

1. **Formular los test correspondientes y adecuados** lo que conllevará :
 - Una implementación de los algoritmos teóricos propuestos.
 - Una descripción y formulación de los test.

⁷Concretamente en la sección de *milestones*, cuya URL es <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales/milestones>.

2. Evaluar los resultados.

El criterio de aceptación de un producto mínimo viable consistirá en:

- La implementación de los algoritmos debe de ser coherente y proveniente del hito anterior y debe de estar referenciada.
- Toda implementación deberá previamente comprobar su correcto funcionamiento mediante tests.
- La redacción del análisis y conclusiones deberá ser aprobado por los tutores nuevamente.

Hito 3: Implementación de aplicaciones reales que se beneficien de las optimizaciones encontradas

A partir de las optimizaciones encontradas del hito previo deberá de implementarse alguna aplicación real que se beneficie de algunas de las optimizaciones encontradas.

Un ejemplo de ello podría ser entrenar una red neuronal en una raspberry pi .

Hito x: Entrega del proyecto

Su resultado será una memoria revisada y pulida que se entregará en Prado como trabajo fin de grado en el plazo de la convocatoria ordinaria.

Para cumplir con los *hitos*, ha sido necesario resolver nuevos problemas menores, estos se han recogido también como *issues* ligados a las historias de usuario y asociados a un *hitos*.

1.4. Registro de trabajo

Con el fin de constatar que el desarrollo efectivamente se ha llevado con una filosofía ágil y según a los tiempos estipulados se ha registrando el tiempo, tareas dedicadas y hitos relacionados en las siguiente [hoja de cálculo](#).

En total el número de horas invertidas ha sido: **TODO añadir registro** Que se distribuye de la siguiente manera en los distintos hitos: **TODO añadir registro**

1.5. Resumen de la metodología

Se ha seguido un desarrollo ágil que radica en la resolución de problemas expuestos como historias de usuarios. Éstas historias de usuario se formulan en base a los beneficiados, personas bien definidas; se formalizan mediante *issues* y entran los *hitos*, hitos con los que guiar el desarrollo.

Todo esto se puede leer no solo en esta memoria sino también en nuestro repositorio de GitHub [[BCC](#)].

1.6. Herramientas utilizadas

1.6.1. GitHub

Como servicio externo hemos usado [GitHub](#), ya que permite implementar de manera eficaz todo el desarrollo ágil: desde la planificación, comunicación, test e incluso difusión y acceso a los resultados.

1.6.2. Lenguaje de programación Julia

Hemos seleccionado como lenguaje de programación [Julia](#), esto se debe a que nos ofrece *benchmarks* muy competitivos⁸, al nivel de C. Así como la disponibilidad de bibliotecas usadas en ciencia de datos de la de lenguajes como *R* y *Python*.

⁸Véase los resultado expuestos en <https://julialang.org/benchmarks/>, web consultada por última vez el 22 de mayo de 2022.

Parte I.
Teoría subyacente

No es usual en un manual que trate sobre redes neuronales encontrarse en su interior con un capítulo sobre teoría de la aproximación, pero tampoco es nuestra intención hacer de este documento una recopilación de todo lo usual, sino todo lo contrario.

Existe en la actualidad un desequilibrio entre resultados empíricos y teóricos de redes neuronales llegando incluso a contradicción (como se comenta en la introducción del capítulo 2), será por tanto nuestro primer objetivo conseguir una revisión y purga de cualquier artificio existente sobre redes neuronales carente de fundamento matemático.

El fin de esto no es más que construir una teoría sólida que de cabida a optimizaciones de fundamento teórico. Para ello nuestro *modus operandi* será el siguiente: Se describirá el conjunto y características de problemas que pretendemos abarcar en el capítulo 2. Al final del mismo capítulo se introduce la definición que hemos determinado por conveniente de red neuronal y que es producto de los capítulos 4 y 3.

Tras todo el fundamento teórico en 5 se explicitará el diseño de la red neuronal así como los algoritmo de evaluación y aprendizaje.

2. Introducción a las redes neuronales

Toma en la actualidad un papel clave en el desarrollo tecnológico el aprendizaje automático [Sar21], siendo las redes neuronales un modelo ampliamente usado en este contexto y por ello nuestro objetivo de democratización.

Comenzaremos pues sentando las bases y formalizando qué significa que una máquina aprenda y cómo puede conseguirse, todo ello en la sección 2.1. Continuaremos en las secciones 4.1 y 5 definiendo las redes neuronales, cómo se construyen y evalúan. **Centraremos además nuestros esfuerzos en el estudio de las redes neuronales de una sola capa**, esto se debe a que a pesar de la tendencia actual de utilizar redes neuronales profundas con un aumento de parámetros que ajustar [BS21] [CZLN21] el sustento de esto no deja de ser experimental o basado en cotas de carácter *en el peor de los casos y por el tamaño del espacio de búsqueda*. Pero estos motivos no constituyen una demostración formal ni rigurosa de porqué decantarnos verdaderamente por ello y es más, otros artículos experimentales demuestran que aumentar el número de capas no mejora los resultado [LGMC21].

Así pues, sustentados con la demostración de convergencia universal [HSW89] de las redes neuronales, la cual asegura la convergencia en igualdad de condiciones, tanto para una capa como para varias y en pos de simplificar el modelo. A priori, para establecer nuestras propias hipótesis de optimización desprovistos de sesgos empíricos, comenzaremos nuestro trabajo tratando tan solo de redes neuronales de una sola capa oculta.

2.1. Concepto de aprendizaje

El término de Aprendizaje Automático [His21] fue acuñado en 1959 por Arthur Samuel para hacer referencia a los sistemas informáticos que pudiesen *aprender* por sí mismos, es decir, mejorar su eficacia y rendimiento de forma autónoma a partir de los datos, sin que en esas mejoras intervenga un programador.

Fue en 1997 cuando Tom Mitchell propuso una definición formal de aprendizaje [Mit97], aproximada a la dada en el libro *Learning from data* [CMo7], que expondremos en seguida.

El aprendizaje es un proceso por el cual se estima una dependencia desconocida (input-output) o la estructura de un sistema a partir de un número finito de observaciones. Se compone de tres elementos principales:

- Un generador o función de distribución de la cual se extraen vectores aleatorios $x \in I \subset \mathbb{R}^n$ que dependen de una función de densidad desconocida ¹.
- Un sistema que produce un vector de salida y por cada entrada del vector x a partir del valor fijo $p(y|x)$, que es desconocido también.
- Una *learning machine* dependiente de parámetros w , que en el caso más general no es más que un conjunto de funciones abstractas cuyos elementos son de la forma $f(x, w)$.

Así pues, el objetivo del aprendizaje automático es encontrar una función que se aproxime a la función de densidad desconocida.

Por lo general la teoría se fundamenta en minimizar el error de estimadores tales como el error cuadrático medio.

2.1.1. Componentes del aprendizaje

A nivel práctico y en nuestro caso, los elementos que consideraremos para el aprendizaje y los cuales serán susceptibles de contribuir a la optimización buscada son (capítulo 1 [AMMIL12]):

- Espacio muestral \mathcal{X} .
- Una función objetivo ideal y desconocida $f : \mathcal{X} \rightarrow \mathcal{Y}$, donde \mathcal{Y} es el conjunto de posibles valores asociados.
- Un conjunto de datos de entrenamiento \mathcal{D} , donde los elementos son pares (x, y) con $x \in \mathcal{X}$ y $y \in \mathcal{Y}$ y que representa una observación $f(x) = y$.
- Un algoritmo de aprendizaje es aquel que dado un espacio de búsqueda o de hipótesis posibles \mathcal{H} con elementos de la forma $h : \mathcal{X} \rightarrow \mathcal{Y}$ con $X \subseteq \mathcal{X}$ y $Y \subseteq \mathcal{Y}$ selecciona un candidato que aproxime f .

En nuestro caso \mathcal{H} será el conjunto de todas las posibles redes neuronales con un cierto número de neuronas y capas ocultas prefijadas.

¹De hecho, encontrar esta función resolvería el problema de aprendizaje

Q En qué consiste la función ideal y desconocida a nivel práctico:

A nivel teórico existen infinitas funciones que coinciden en un conjunto finito de puntos, por lo que verdaderamente no se está buscando una función ideal sino un elemento dentro de la clase de equivalencia de las funciones medibles que coincide en un conjunto numerable de puntos.

2.1.2. Tipos de aprendizaje

El aprendizaje a partir de los datos trata en esencia de modelizar un patrón o ley del proceso subyacente a partir de un conjunto finito de muestras. En función de ciertas características del conjunto de datos se tienen distintos tipos de aprendizaje y cabe la pregunta de en qué ámbitos son útiles las redes neuronales o si resultados en problemas concretos son extensibles a otros.

Con el fin de tener una clara distinción, establecemos la siguiente clasificación sobre los tipos de aprendizajes existentes que se basada en el artículo [Sar21], el capítulo primero de [AMMIL12] y el capítulo cuarto página 179 de [Biso6].

2.1.2.1. Aprendizaje supervisado

Cuando el conjunto de datos de entrenamiento contiene de manera explícita lo que es una salida correcta respecto a una entrada estamos frente a un caso de **aprendizaje supervisado**. Este tipo de aprendizaje resuelve tareas de clasificación, donde la salida es discreta o de regresión, con una salida continua. Un ejemplo sería el reconocimiento de dígitos manuscritos donde cada imagen de un dígito tiene asociado cuál es.

2.1.2.2. Aprendizaje por refuerzo

Se trata de un problema de aprendizaje por refuerzo cuando conjunto de datos de entrenamiento no explicita la salida, en su lugar contiene posibles salidas junto a la bondad de éstas.

Pongamos por ejemplo que se quiere enseñar a un sistema a jugar a las damas; de todo el espacio de jugadas posibles, se conocerían tan solo el resultado de algunas situaciones, por ejemplo en la que uno de los jugadores ha ganado.

2.1.2.3. Aprendizaje no supervisado

En este tipo de aprendizaje, los datos de entrenamiento tampoco contienen ninguna información de la salida. Tan solo se tienen los datos de entrada. El **aprendizaje no supervisado** consiste en la tarea de encontrar patrones y estructuras en los datos de entrada, así como de crear una abstracción de los datos. Son problemas de *clustering*, asociación o entre otros, detección de anomalías.

2.1.2.4. Aprendizaje semi supervisado

Combina tanto datos etiquetados como no etiquetados, se utiliza en problemas de clasificación o *clustering* como por ejemplo en traducción o detección de fraudes.

Las redes neuronales son partícipes en los tres tipos de aprendizaje recién mencionados [DZDW18], [BGNR16], [SM02]. Sin embargo centraremos nuestro estudio en el caso de aprendizaje supervisado.

3. Teoría de la aproximación

Puesto que queremos fundamentar desde el origen las redes neuronales, supongamos que desconociéramos la existencia de éstas y nos halláramos frente a un problema de aprendizaje 2.1; el enfoque más natural consistiría en abordar el problema haciendo uso de teoría de la aproximación. ¿Somos capaces de aproximar una función a partir de sus puntos?

3.1. Polinomios de Bernstein

En esta sección introduciremos los polinomios de Bernstein, que vistos como una serie nos asegurarán una convergencia uniformemente a cualquier función continua en un compacto y serán esenciales para nuestra prueba del teorema de Stone-Weierstrass, las pruebas se basan en el manual [Bar47].

Comenzaremos recordando el Teorema del Binomio de Newton.

Teorema 3.1 (Binomio de Newton). *Cualquier potencia de un binomio $x + y$ con $x, y \in \mathbb{R}$, puede ser expandido en una suma de la forma*

$$(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k}.$$

Tomando ahora para esta igualdad $x \in \mathbb{R}, y = 1 - x$ se tiene que

$$1 = (x + (1 - x))^n = \sum_{k=0}^n \binom{n}{k} x^k (1 - x)^{n-k}. \tag{3.1}$$

Dada cualquier función continua $f : I \rightarrow \mathbb{R}$ podríamos multiplicar la ecuación (3.1) por $f(x)$ resultando

$$f(x) = \sum_{k=0}^n f(x) \binom{n}{k} x^k (1 - x)^{n-k}. \tag{3.2}$$

Y tomando como dominio $I = [0, 1]$ de $f : I \rightarrow \mathbb{R}$, nos encontramos frente a una ecuación muy sugerente para introducir $B_n(x)$, el *Polinomio n -ésimo de Bernstein*. El cual pretende aproximar la función f a través de los puntos $\frac{k}{n}$ con $n \in \mathbb{N}$ fijo y $k \in \{0, \dots, n\}$.

Definición 3.1 (Polinomios de Bernstein). Dada cierta función $f : [0, 1] \rightarrow \mathbb{R}$, se define el n -ésimo polinomio de Bernstein para f como

$$B_n(x) = B_n(x; f) = \sum_{k=0}^n \left(f\left(\frac{k}{n}\right) \binom{n}{k} x^k (1 - x)^{n-k} \right).$$

Faltaría por ver si efectivamente nuestro polinomio construido *aproxima lo suficientemente bien* a la f originaria. Basándonos en la igualdad (3.2) y la diferencia entre $f(x)$ y $B_n(x)$ se concluye que

Ⓢ ¿Qué es la convergencia uniforme?. Cuando se hable de **convergencia** nos referiremos a encontrar un elemento que **aproxime todo lo bien que queramos**; es decir, fijado cualquier error podemos encontrar un elemento que lo aproxime cometiendo un error menor.

Tengamos presente que en el contexto de funciones el error se puede medir evaluando la imagen de cada punto del dominio; entonces con **uniforme** lo que se dice es que independientemente del error que se busque, se puede encontrar una función que en **para todos los puntos del dominio aproxima al objetivo por debajo de ese error**.

Ⓢ ¿Con qué idea se está avanzando?.

Nuestro objetivo es ser capaces de aproximar una función a partir de una muestra de sus preimágenes e imágenes. Comenzaremos probando primero que dada una función continua, podemos aproximarla a partir de una muestra concreta. Por eso se está tratando de reescribir la función en términos de cierto conjunto de sus imágenes.

Ⓢ **Idea intuitiva compacto**
Podría entenderse compacto como una forma de indicar que todos los valores posibles que puede tomar la entrada están dentro de un rango.

3. Teoría de la aproximación

$$|f(x) - B_n(x)| \leq \sum_{k=0}^n \left| f(x) - f\left(\frac{k}{n}\right) \right| \binom{n}{k} x^k (1-x)^{n-k} \quad (3.3)$$

Ante tal igualdad la intuición ya nos hace pensar que sea convergente al aumentar el tamaño de la partición. En efecto, en nuestro sucesivo teorema, nos cercioraremos que 3.1 es uniformemente convergente a f en un compacto.

Para la demostración será necesario el siguiente resultado técnico:

Lema 3.1. *Se desea probar la siguiente igualdad*

$$\frac{1}{n} x(1-x) = \sum_{k=0}^n \left(x - \frac{k}{n}\right)^2 \binom{n}{k} x^k (1-x)^{n-k}. \quad (3.4)$$

Ⓢ **Resultado técnico.**

La demostración del lema 3.4 se ha incluido para que todo resultado esté fundamentado, solo se utiliza para establecer una cota necesaria en una desigualdad del teorema 3.2.

Demostración. Tengamos ahora presente las siguientes igualdades

$$\binom{n-1}{k-1} = \frac{(n-1)!}{(k-1)!(n-1-(k-1))!} = \frac{k}{n} \binom{n}{k} \quad (3.5)$$

$$\binom{n-2}{k-2} = \frac{(n-2)!}{(k-2)!(n-2-(k-2))!} = \frac{k(k-1)}{n(n-1)} \binom{n}{k} \quad (3.6)$$

Partiendo de la igualdad (3.1):

$$1 = (x + (1-x))^n = \sum_{k=0}^n \binom{n}{k} x^k (1-x)^{n-k}.$$

Reemplazamos la n por $n-1$ y la k por j y tenemos

$$1 = \sum_{j=0}^{n-1} \binom{n-1}{j} x^j (1-x)^{(n-1)-j}.$$

Multiplicamos por x y aplicamos la igualdad (3.5) resultando

$$x = \sum_{j=0}^{n-1} \frac{j+1}{n} \binom{n}{j+1} x^{j+1} (1-x)^{n-(j+1)}.$$

Renombramos $k = j + 1$, por lo que resulta

$$x = \sum_{k=1}^n \frac{k}{n} \binom{n}{k} x^k (1-x)^{n-k}.$$

Como el término con $k = 0$ es nulo podemos añadirlo a la sumatoria

$$x = \sum_{k=0}^n \frac{k}{n} \binom{n}{k} x^k (1-x)^{n-k}. \quad (3.7)$$

Haremos ahora un razonamiento similar sustituyendo n por $n-2$. Partiendo de (3.1) se tiene que

$$1 = (x + (1-x))^n = \sum_{k=0}^n \binom{n}{k} x^k (1-x)^{n-k}.$$

Reemplazamos la n por $n - 2$ y la k por j y tenemos

$$1 = \sum_{j=0}^{n-2} \binom{n-2}{j} x^j (1-x)^{(n-2)-j}.$$

Multiplicamos por x^2 y aplicamos la igualdad (3.6) resultando

$$x^2 = \sum_{j=0}^{n-2} \frac{(j+2)(j+1)}{n(n-1)} \binom{n}{j+2} x^{j+2} (1-x)^{n-(j+2)}.$$

Renombramos $k = j + 2$, por lo que resulta

$$x^2 = \sum_{k=2}^n \frac{k(k-1)}{n(n-1)} \binom{n}{k} x^k (1-x)^{n-k}.$$

Como con los términos $k = 0$ y $k = 1$ se anula, podemos añadir dichos índices sin modificar la suma

$$x^2 = \sum_{k=0}^n \frac{k(k-1)}{n(n-1)} \binom{n}{k} x^k (1-x)^{n-k}.$$

Podemos reescribir la ecuación resultando :

$$(n^2 - n)x^2 = \sum_{k=0}^n (k^2 - k) \binom{n}{k} x^k (1-x)^{n-k}.$$

Sumando las dos expresiones que hemos obtenido (3.7) y (3.1) resultando

$$(n^2 - n)x^2 + nx = \sum_{k=0}^n ((k^2 - k) + k) \binom{n}{k} x^k (1-x)^{n-k}.$$

Dividimos todo entre n .

$$\left(1 - \frac{1}{n}\right)x^2 + \frac{1}{n}x = \sum_{k=0}^n \left(\frac{k}{n}\right)^2 \binom{n}{k} x^k (1-x)^{n-k}. \quad (3.8)$$

A continuación sumamos a la igualdad (3.8) la ecuación (3.1) multiplicada por x^2 y la ecuación (3.7) multiplicada por $-2x$ resultando:

$$\left(1 - \frac{1}{n} + 1 - 2\right)x^2 + \frac{1}{n}x = \sum_{k=0}^n \left(\left(\frac{k}{n}\right)^2 + x^2 - 2x\right) \binom{n}{k} x^k (1-x)^{n-k}.$$

Sacando factor común en el miembro de la izquierda y escribiendo como un cuadrado el factor de la derecha, resulta la igualdad 3.4 que buscábamos

$$\frac{1}{n}x(1-x) = \sum_{k=0}^n \left(x - \frac{k}{n}\right)^2 \binom{n}{k} x^k (1-x)^{n-k}.$$

□

Teorema 3.2 (Teorema de aproximación de Bernstein). *Sea f una función continua de un intervalo $I = [0, 1]$ con imágenes en los reales. La secuencia de polinomio de Bernstein 3.1 converge uniformemente a f en I .*

3. Teoría de la aproximación

Recordaremos antes la definición de convergencia uniforme:

Definición 3.2 (Convergencia uniforme para funciones reales). Dado E un conjunto y $\{f_n\}_{n \in \mathbb{N}}$ una sucesión de funciones de E a los reales, se dice que dicha sucesión converge uniformemente si para cualquier $\varepsilon > 0$ existe un número natural m tal que para todo $x \in E$ y cualquier natural n que cumpla $n \geq m$ se tiene que

$$|f_n(x) - f(x)| < \varepsilon$$

Comencemos pues con la demostración del teorema 3.2.

Demostración. Para cualquier $\varepsilon > 0$ queremos probar que existe un $m_\varepsilon \in \mathbb{N}$ tal que para todo $x \in I$ e $n \geq m_\varepsilon$ se tiene que $|f(x) - B_n(x)| < \varepsilon$.

Para ello por estar f definida en un intervalo, se tienen dos consecuencias claves:

1. Está acotada, supongamos por $M \in \mathbb{R}$, esto es $|f(x)| \leq M$.
2. En virtud del teorema de Heine-Cantor f es uniformemente continua, es decir, por estar f definida en un compacto, para cualquier $\varepsilon > 0$ existirá un δ_ε tal que para cualesquiera $x, y \in I$ que cumplan $|x - y| < \delta_\varepsilon$ entonces $|f(x) - f(y)| < \varepsilon$.

En virtud de la consecuencia 1. Dado $N \in \mathbb{N}$ fijo pero arbitrario, para cualquier $k \in \{1, \dots, N\}$ se tiene que $\frac{k}{N} \in I$ y tomando $x \in I$ podemos acotar por la desigualdad triangular

$$\left| f(x) - f\left(\frac{k}{N}\right) \right| \leq |f(x)| + \left| f\left(\frac{k}{N}\right) \right| \leq 2M.$$

Por lo que

$$|f(x) - B_n(x)| \sum_{k=0}^n \left| f(x) - f\left(\frac{k}{n}\right) \right| \leq \binom{n}{k} x^k (1-x)^{n-k}.$$

Puesto que tenemos que f acotada por M y es uniformemente continua, para valores de k tales que $\frac{k}{n}$ esté próxima a x , tal término de la sumatoria será pequeño por la continuidad de f en x . Por otro lado si está lo suficientemente alejado, tan solo podremos acotar tal término por $2M$.

Para $\varepsilon > 0$ y para δ_ε de la definición de continuidad uniforme de f podemos encontrar un

$$n \geq \sup \left\{ (\delta_\varepsilon)^{-4}, \frac{M^2}{\varepsilon^2} \right\}. \quad (3.9)$$

Separaremos pues nuestra sumatoria en los siguientes dos conjuntos.

$$\mathcal{A}_{nx} = \left\{ k \text{ tales que } k \in \{0, \dots, n\} \text{ y } \left| x - \frac{k}{n} \right| < n^{-\frac{1}{4}} \leq \delta_\varepsilon \right\},$$

$$\mathcal{B}_{nx} = \{0, \dots, n\} - \mathcal{A}_{nx}.$$

Para los elementos de \mathcal{A}_{nx} se obtiene la siguiente estimación:

$$\begin{aligned} \sum_{k \in \mathcal{A}} \left| f(x) - f\left(\frac{k}{n}\right) \right| \binom{n}{k} x^k (1-x)^{n-k} &\leq \sum_{k \in \mathcal{A}_{nx}} \varepsilon \binom{n}{k} x^k (1-x)^{n-k} = \\ &= \varepsilon \sum_{k \in \mathcal{A}_{nx}} \binom{n}{k} x^k (1-x)^{n-k} \leq \varepsilon \sum_{k=0}^n \binom{n}{k} x^k (1-x)^{n-k} = \varepsilon \end{aligned}$$

Para el resto de sumandos para los que $|x - \frac{k}{n}| \geq n^{-\frac{1}{4}}$ se tiene que $(x - \frac{k}{n})^2 \geq n^{-\frac{1}{2}}$ cota. Procederemos a acotar tales términos, para ello partimos de la ecuación (3.3) y la acotación de f .

$$\begin{aligned} |f(x) - B_n(x)| & \sum_{k=0}^n \left| f(x) - f\left(\frac{k}{n}\right) \right| \leq \sum_{k=0}^n 2M \binom{n}{k} x^k (1-x)^{n-k} \\ & \sum_{k \in \mathcal{B}_{nx}} 2M \binom{n}{k} x^k (1-x)^{n-k} \\ & = 2M \sum_{k \in \mathcal{B}_{nx}} \frac{\left(x - \frac{k}{n}\right)^2}{\left(x - \frac{k}{n}\right)^2} \binom{n}{k} x^k (1-x)^{n-k} \\ & \leq 2M \sqrt{n} \sum_{k \in \mathcal{B}_{nx}} \left(x - \frac{k}{n}\right)^2 \binom{n}{k} x^k (1-x)^{n-k} \\ & \leq 2M \sqrt{n} \left(\frac{1}{n} x(1-x)\right) \leq \frac{M}{2\sqrt{n}}, \end{aligned}$$

Donde para la penúltima desigualdad se ha usado la desigualdad (3.4) del lema previo y para la última que en el intervalo I se satisface que

$$x(1-x) \leq \frac{1}{4}.$$

Además recordemos que se puede tomar un n que satisfaga (3.9) y entonces se concluye que para los valores de \mathcal{B}_{nx} se puede acotar la desigualdad por ε .

Por tanto para un n convenientemente seleccionado, se ha acotado la desigualdad para los índices \mathcal{A}_{nx} y \mathcal{B}_{nx} , es decir todos, por lo que concluimos que

$$|f(x) - B_n(x)| \leq 2\varepsilon,$$

independiente del valor de x , por lo que se prueba que la secuencia de polinomios de Bernstein converge uniformemente a f en I . \square

Realizando un repaso global habiendo acabado el teorema, se pueden extraer que conjunto a un ingenioso manejo de operaciones y acotaciones; la clave del resultado reside en las consideraciones en 1 y 2 y estas a su vez en la compacidad de I .

Por su parte, la selección del dominio de $I = [0, 1]$ viene determinada ya que los nodos $\{\frac{k}{N} | k \in \{0, \dots, N\}\}$ sobre los que se construye el N -ésimo polinomio de Bernstein deben pertenecer a I .

Sin embargo, tal dificultad es fácilmente salvable con un homeomorfismo.

Como resultado de relajar el dominio donde se define f , pidiéndole tan solo compacidad nace el siguiente corolario.

Corolario 3.1 (Teorema de aproximación de Weierstass). *Sea f una función continua definida en un intervalo real y con valores reales. Se tiene que f puede ser aproximada uniformemente con polinomios.*

② ¿Qué es un homeomorfismo?

A nivel intuitivo un homeomorfismo sería que una transformación de un objeto que lo desforma *sin romperlo*. Es decir, si este fuera plastilina sería aquellas figuras que podríamos llegar sin separar la masilla o crear y eliminar agujeros.

3. Teoría de la aproximación

Demostración. Si f se encuentra definida en $[a, b]$ con $a < b$ y bastará considerar la función

$$g(t) = f((b-a)t + a) \text{ con } t \in [0, 1]$$

Esta función está definida en $[0, 1]$, tiene la misma imagen que f y mantiene la continuidad ya que se ha construido a través del homeomorfismo $H : [0, 1] \rightarrow [a, b]$, con $H(t) = (b-a)t + a$.

En virtud del teorema de convergencia 3.2 g podrá ser aproximada uniformemente por una sucesión de polinomios de Bernstein $\{S_n\}_{n \in \mathbb{N}}$. Gracias a ella se puede construir la sucesión $\{S_n \circ H^{-1}\}_{n \in \mathbb{N}}$, que aproxima uniformemente a f . \square

3.2. Teorema de Stone-Weierstrass

Teorema 3.3 (Teorema de Stone-Weierstrass). *Sea K un subconjunto compacto de \mathbb{R}^p y sea \mathcal{A} una colección de funciones continuas de K a \mathbb{R} cumpliendo \mathcal{A} y que separa puntos en K , es decir cumpliendo las siguientes propiedades:*

1. *La función constantemente uno, definida como $e(x) = 1$, para cualquier $x \in K$ pertenece a \mathcal{A} .*
2. *Cerrado para sumas y producto para escalares reales. Si f, g pertenece a \mathcal{A} , entonces $\alpha f + \beta g$ pertenece a \mathcal{A} .*
3. *Cerrado para producto. Para $f, g \in \mathcal{A}$, se tiene que fg pertenece a \mathcal{A} .*
4. *Separación de K , es decir si $x \neq y$ pertenecientes a K , entonces existe una función f en \mathcal{A} de tal manera que $f(x) \neq f(y)$.*

Se tiene que toda función continua de K a \mathbb{R} puede ser aproximada en K por funciones de \mathcal{A} .

La idea que subyace bajo la demostración del teorema es que a partir de las hipótesis de la estructura algebraica y separabilidad se prueba que es posible encontrar las funciones máximo y mínimo, construyéndolo a partir del valor absoluto.

Con esta nueva propiedad y siendo cerrado para combinaciones lineales se puede aproximar uniformemente cualquier función continua definida en el compacto.

Demostración. Sean $a, b \in \mathbb{R}$ y $x \neq y$ pertenecientes a K . Por la hipótesis de separabilidad existirá una función $f \in \mathcal{A}$ tal que $f(x) \neq f(y)$. Además la existencia de un elemento neutro $e \in \mathcal{A}$ en el álgebra nos permite encontrar reales α, β tales que

$$\alpha f(x) + \beta e(x) = a, \quad \alpha f(y) + \beta e(y) = b$$

Por el teorema de Heine, para $f \in \mathcal{A}$ está acotada por tomar imagen en un compacto, es decir $|f(x)| \leq M$ para $x \in K$.

Consideremos ahora la función valor absoluto, $\phi(t) = |t|$ definida en el dominio $I = [-M, M]$. Por el teorema de aproximación de Weierstrass 3.1 para cualquier $\varepsilon > 0$ existirá un polinomio p cumpliendo que

$$||t| - p(t)| < \varepsilon, \quad \forall t \in I.$$

Puesto que $t \in I$ no son más que las posibles imágenes que puede tomar f en K inferimos entonces que

$$|f(x) - p \circ f(x)| < \varepsilon \quad \forall x \in K.$$

Como $f \in \mathcal{A}$ y p es un polinomio, es decir, $p \circ f(x)$ son sumas de potencias multiplicadas por escalares de $f(x)$, luego por la hipótesis de ser cerrado a estas operaciones tenemos que la función $|f|$ pertenece a \mathcal{A} si $f \in \mathcal{A}$.

Tenemos con esto que \mathcal{A} también es cerrada a supremo e ínfimo gracias a que:

$$\sup\{f, g\} = \frac{1}{2}\{f + g + |f + g|\}$$

$$\inf\{f, g\} = \frac{1}{2}\{f + g - |f + g|\}$$

Por lo que concluimos que cualquier función puede ser uniformemente aproximada por combinaciones lineales, supremos e ínfimo mediante funciones de \mathcal{A} . Es decir, cualquier función continua en K puede ser uniformemente aproximado por funciones de \mathcal{A} . \square

3.3. Conclusiones teoría de la aproximación

Acabamos de probar en 3.2 que cualquier función continua es aproximable uniformemente con polinomios en un compacto. Sin embargo este enfoque tiene los siguientes problemas:

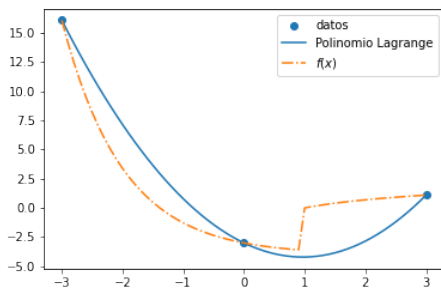
1. La prueba obtenida no nos permite una forma constructiva sencilla de obtener el polinomio.
2. Aproximando con polinomios se corre el riesgo de que fuera de la muestra el error sea demasiado grande. Como muestra de ello damos un ejemplo patológico reflejado en la figura 3.1; se pretende aproximar la función $f : [-3, 3] \rightarrow \mathbb{R}$ definida como

$$f(x) = \begin{cases} e^{-x} + 4 & \text{si } x \leq 1 \\ \log x & \text{si } x > 1 \end{cases}$$

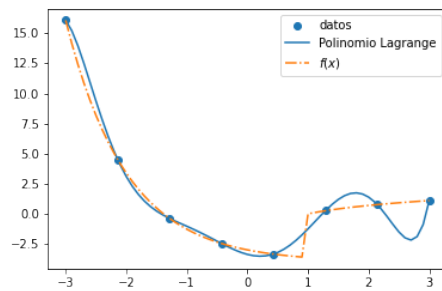
usando el método de interpolación de Lagrange; en este caso el error tiende a infinito conforme aumenta el número de nodos.¹

3. Y otra cuestión, de corte físico o filosófico ¿son todos los fenómenos observables continuos? Sería extraño que así fueran, lo que evidencia la necesidad de formular una teoría más general.

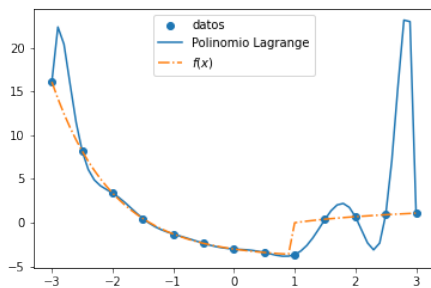
¹Puede encontrar la implementación del código que genera las gráficas en nuestro repositorio <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales>, en el fichero Lagrange.ipynb que se encuentra en el directorio de teoría de la aproximación de la memoria.



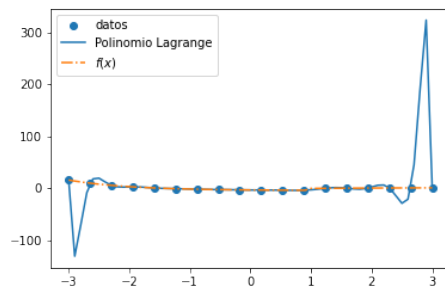
(a) Polinomio de Lagrange utilizando 3 datos



(b) Polinomio de Lagrange utilizando 8 datos



(c) Polinomio de Lagrange utilizando 13 datos



(d) Polinomio de Lagrange utilizando 18 datos

Figura 3.1.: Ejemplo de aproximación de la función $f(x)$ a partir de los polinomios de Lagrange.

El problema que evidencia este caso patológico es el tratar de abarcar todo el dominio con un mismo polinomio ¿y si en lugar de eso se hicieran aproximaciones en una partición concreta del dominio? El resultado sería una función definida a trozos. La cuestión es que esta aproximación sería difícil de implementar de manera eficiente; sin embargo, es el germen y el enfoque de las *funciones de activación*.

De todas formas no abandonemos del todo esta teoría, porque como ya veremos el teorema de Stone Weierstrass 3.2 jugará un papel fundamental es la demostración de que las redes neuronales son aproximadores universales.

Las funciones de activación Γ son la clave del aprendizaje

Las *funciones de activación* serán definidas con profundidad en la sección 4.2, pero para continuar con nuestro razonamiento pensemos en ellas como una función cualquiera que no sea un polinomio.

Una vez liberados de tratar de buscar un polinomio que aproxime la función en todo el dominio, podemos pensar en aproximar la imagen de acorde a intervalos.

TODO : Añadir gráficos cuando esté implementada una red neuronal

La idea intuitiva es que para una capa oculta con una neurona, lo que se hace es *colocar* por escalado y simetrías la imagen de la función de activación.

3. Teoría de la aproximación

Una red neuronal de dos nodos en la capa oculta sería $(h_1(x) + h_2(x)) = h \in \mathcal{H}_{1,2,1}$ y si por ejemplo

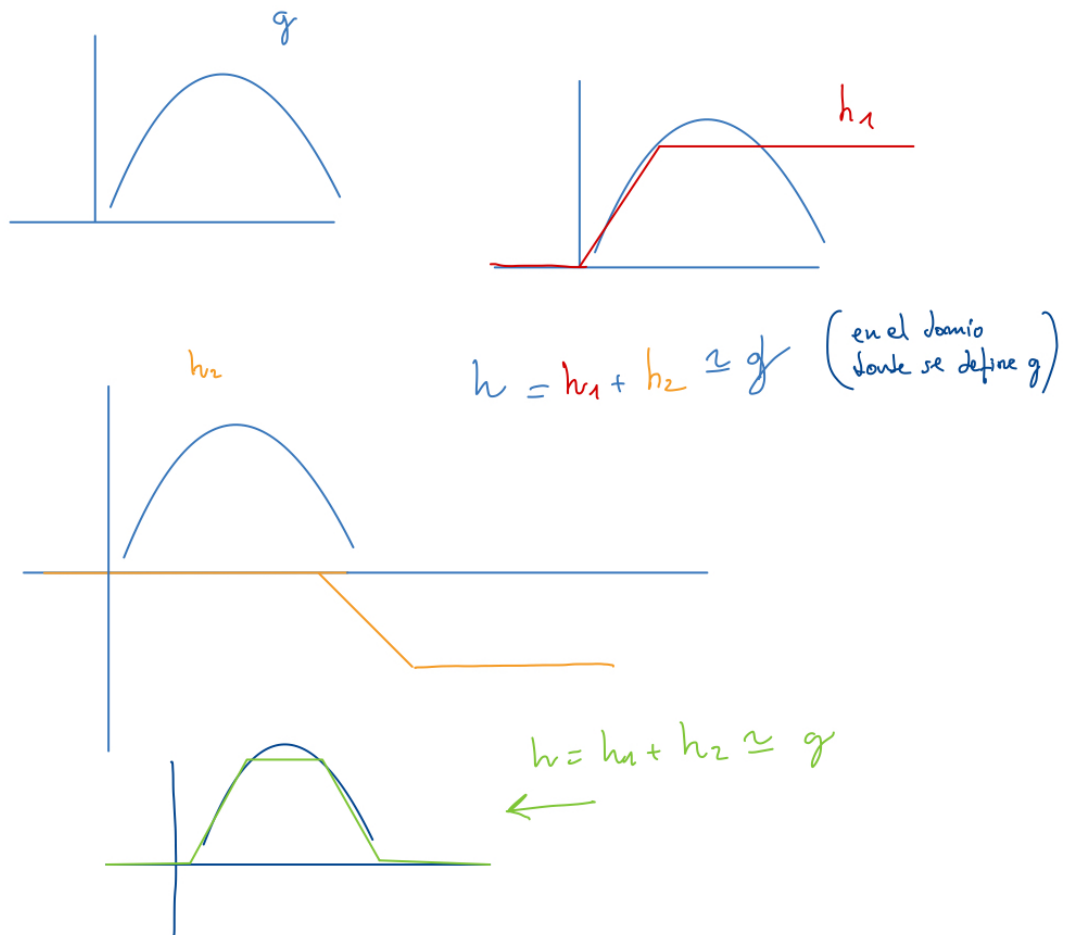
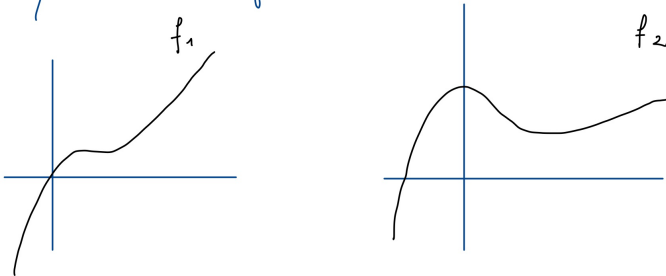


Figura 3.2.: Cómo actúa en la aproximación una función de activación

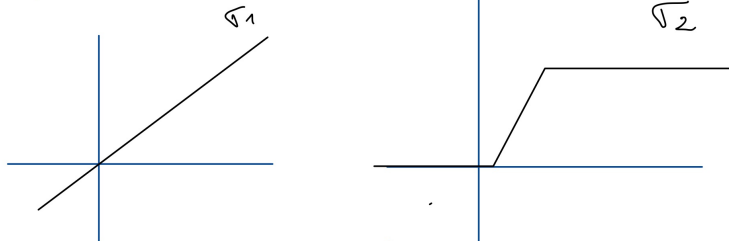
La importancia función activación
adaptándose datos.

Ejemplos obvios tras lo visto:

Si solo tuviéramos una neurona
y nuestras funciones objetivo fueran:



Funciones activación



Está claro que con transformación
elemental σ_1 aproxima mejor f_1 y
 f_2 aproxima mejor σ_2 .

Figura 3.3.: Cómo afecta la forma de la función de activación

4. Las redes neuronales son aproximadores universales

4.1. Definición de las redes neuronales *Feedforward Networks* de una capa oculta

Ya estamos en condiciones de introducir qué es una red neuronal cómo está construida y en qué consiste el *aprendizaje* de la misma. Concretamente construiremos el tipo particular *Feedforward Neural Networks*, al cual nos referiremos de ahora en adelante como red neuronal.

De acorde con nuestra filosofía de trabajo expuesta en la introducción del capítulo 2 partiremos de un modelo de una sola capa oculta y además comenzaremos presentando el modelo que hemos convenido más conveniente, esta decisión es argumentada en el capítulo 5.

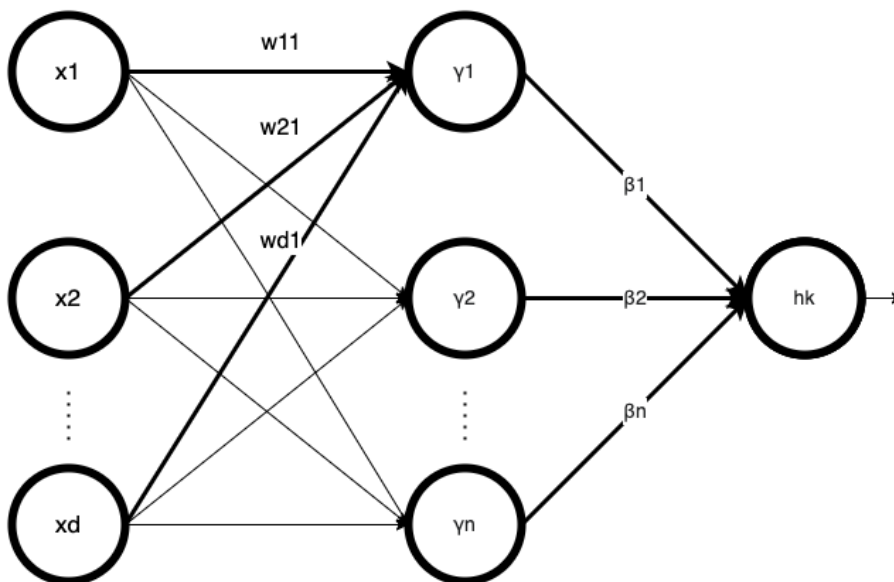


Figura 4.1.: Grafo de una red neuronal de una capa oculta

Aportación original

Definición 4.1 (Redes neuronales de una capa oculta). Dados $X \subseteq \mathbb{R}^d, Y \subseteq \mathbb{R}^s$ y Γ un

4. Las redes neuronales son aproximadores universales

conjunto no vacío de funciones medibles definidas de \mathbb{R} a \mathbb{R} , denotaremos como

$$\mathcal{H}(X, Y) = \{h : X \rightarrow Y / h_k(x) = \sum_{i=1}^n \beta_{ik} \gamma_i(A_i(x)),$$

donde h_k es la proyección k -ésima de h con $k \in \{1, \dots, s\}$,

$n \in \mathbb{N}$, $\gamma_i \in \Gamma$, $\beta_{ik} \in \mathbb{R}$ y A_i una aplicación afín de \mathbb{R}^d a \mathbb{R} .

② Interpretación fórmula

Observemos que n es el número de neuronas de la capa oculta. Es decir lo que en el grafo 4.1 serían las neuronas de las capas ocultas se correspondería con términos $\gamma_i(A_i(x))$.

Es habitual representar una red neuronal de forma matricial, veremos que tal forma es equivalente a la definición dada.

Consideramos la aplicación inclusión $i : \mathbb{R}^r \rightarrow \mathbb{R}^{r+1}$ dada por $i((x_1, \dots, x_d)) = (1, x_1, \dots, x_d)$. Para coeficientes $w_i \in \mathbb{R}$ toda función afín es de la forma $A_i(x) = \sum_{j=1}^d (w_{ji} x_j) + w_{0i}$, tomando $W_i = (w_{0i}, w_{1i}, \dots, w_{ri}) \in \mathbb{R}^{r+1}$ tenemos que $A_i(x) = W_i \cdot i(x)$ como queríamos probar.

Además también se suele mostrar de manera pedagógica con un grafo como el mostrado en la figura 4.1.

Componentes de la red neuronal

A la vista de la definición dada notemos que cada elemento de $\mathcal{H}(X, Y)$ viene determinado por los coeficientes de las distintas β y w s de la función afín. Como veremos más adelante estos son los valores que *aprende una red neuronal*.

Diferencia con otras definiciones

En otros textos como en el capítulo cinco, páginas 227-256 del libro [Biso06] y las notas online sobre redes neuronales de [AMMIL12] se presentan las redes neuronales de una capa como

$$y_k(x, w) = \theta_k \left(\sum_{j=1}^M w_{ji}^{(2)} \sigma_j \left(\sum_{i=1}^D w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \\ = \theta_k \left(\sum_{j=1}^M A_k^{(n_k)} \left(\sigma \left(A_{jk}^r(x) \right) \right) \right) \text{ para cada } k \in \{1, \dots, K\}.$$

Donde θ_k representa una función de *clasificación*, σ_j lo que se suele llamar *función de activación* y a la que además se le exige que sea diferenciable.

Las diferencias con nuestra definición son las siguientes

- **Desaparece la función de clasificación θ .** El motivo es que es un artificio teóricamente innecesario de acorde al teorema de convergencia universal 4.1.
- **Se elimina un parámetro** de la transformación afín de la última capa; puesto que no es necesario para la convergencia de nuevo por 4.1 lo hemos eliminado.
- Nuestras funciones de activación son funciones medibles en vez de diferenciables ya que a priori no existe ninguna hipótesis teórica que fuerce a tal restricción, como hemos visto en 4.1.

② Qué son las funciones medibles y porqué las usamos en nuestra definición. A nivel intuitivo una función medible es aquella, que por muy extraña que sea, su imagen (los valores que toma su salida) está acotada casi siempre, lo que a nivel práctico significa que podemos observar y cuantificar sus valores. Con esto pretendemos que nuestra definición Γ sea lo menos restrictiva posible.

4.2. Las redes neuronales son aproximadores universales

Tras la definición 4.1 de red neural expuesta, es pertinente la pregunta si tal estructura será capaz de aproximar con éxito una función genérica desconocida.

Aunque las redes neuronales multicapa ya se venían aplicando con anterioridad, véase por ejemplo los usos expuestos durante la primera conferencia internacional de redes neuronales de [43087] de 1987, no fue hasta 1989 que se descubrió formalmente su alcance. Tal delimitación se propuso en el artículo **Multilayer Feedforward Networks are Universal Approximators** [HSW89] escrito por Kurt Hornik, Maxwell Stinchcombe y Halber White enunciando:

Teorema 4.1. *Las redes feedforward multicapa son una clase de aproximadores universales. Una red neuronal feedforward multicapa estándar con tan solo una capa oculta y con una función de activación cualquiera es capaz de aproximar cualquier función Borel medible con dominios y codominios de dimensión finita (no necesariamente iguales) y con el nivel de precisión que se desee siempre y cuando se utilicen suficientes neuronas. En este sentido las redes feedforward multicapa son una clase de aproximadores universales.*

En las secciones siguientes, con el fin de alcanzar una comprensión profunda de las redes neuronales, trataremos de desgranar y profundizar en el artículo y su demostración. Primero precisaremos o introduciremos conceptos elementales sobre redes neuronales 4.3, después demostraremos el teorema en el caso real 4.2 e iremos refinando y generalizando los resultados hasta probar el resultado enunciado 4.1 para una capa oculta.

El esquema general será:

- Las redes neuronales que nosotros hemos modelizado son densas en un espacio más general que hemos denominado *Anillo de aproximación de redes neuronales*.
- El espacio *Anillo de aproximación de redes neuronales* es denso en la funciones continuas.
- Las funciones continuas son densas en los reales.

Si quisiéramos situar en este esquema a otras definiciones de redes neuronales las situaríamos entre nuestro modelo y el espacio *Anillo de aproximación de redes neuronales*; en el capítulo 5 se probará la equivalencia de nuestro modelo con estas definiciones y sus beneficios.

© Idea intuitiva conjunto denso. Si S es denso en T , se está diciendo que los elementos de S son capaces de aproximar cualquier elemento de T con la precisión que se desee.

4.3. Definiciones primeras

Comenzaremos presentando definiciones básicas sobre redes neuronales.

Definición 4.2 (Función de activación). Una función $\psi : \mathbb{R} \rightarrow [0, 1]$ es una **función de activación** si cumple las siguientes propiedades:

- (I) Es no decreciente.
- (II) $\lim_{x \rightarrow \infty} \psi(x) = 1$.
- (III) $\lim_{x \rightarrow -\infty} \psi(x) = 0$.

Ejemplos comunes de funciones de activación son

Q Observación sobre la idoneidad de cada función de activación: Se probará la convergencia de las redes neuronales independientemente de la función de activación seleccionada. Cabe entonces la pregunta ¿Existen funciones de activación más democráticas que otras? Se discutirá esta pregunta en 6.

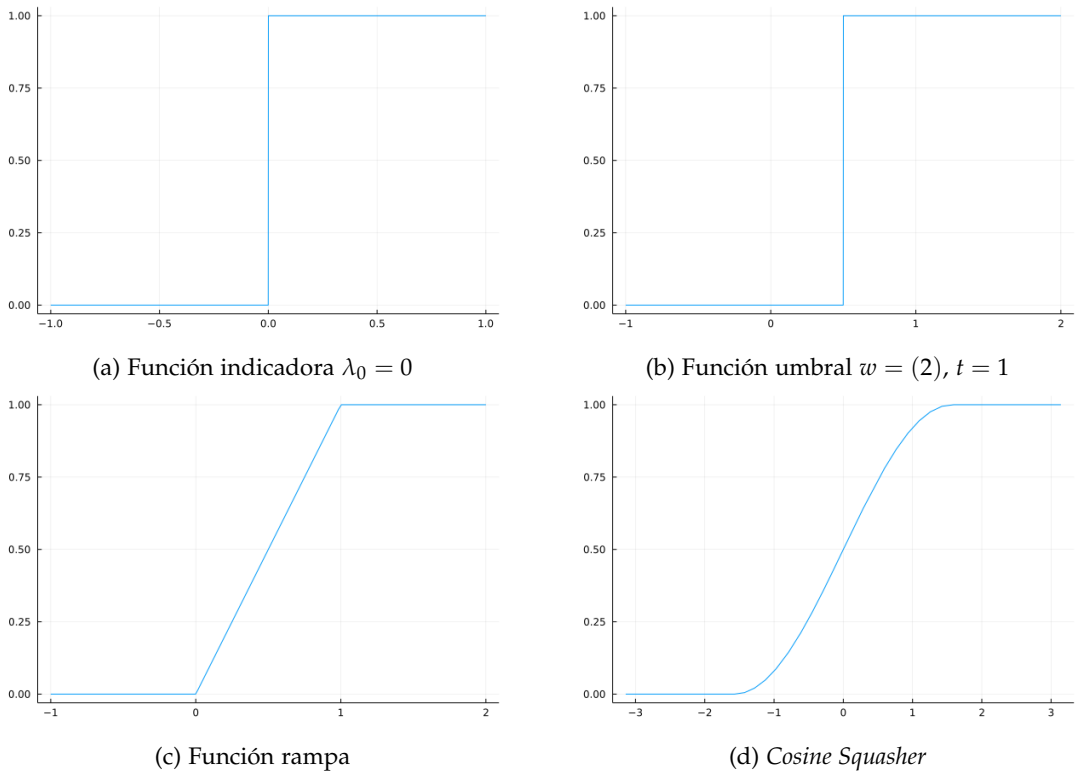


Figura 4.2.: Ejemplos de funciones de activación

- **Funciones indicadoras 4.2a:** $\psi(\lambda) = 1_{\{\lambda > \lambda_0\}}$ con $\lambda_0 \in \mathbb{R}$.
- **Funciones umbral 4.2b:** Una función umbral, es una función booleana monótona $\psi_w : \{0, 1\}^n \rightarrow \{0, 1\}$, donde para $w \in \mathbb{R}^n, t \in \mathbb{R}$ fijos se satisface que

$$\psi_w(x) = \begin{cases} 1, & \text{si } w \cdot x \geq t \\ 0, & \text{si } w \cdot x < t. \end{cases}$$

- **Función rampa 4.2c:** $\psi(\lambda) = \lambda 1_{\{0 \leq \lambda \leq 1\}} + 1_{\{\lambda > 1\}}$.

- La función *cosine squasher* de Gallant and White 4.2d (1988) [GW88].

$$\psi(\lambda) = \left(1 + \cos\left(\lambda + 3\frac{\pi}{2}\right)\frac{1}{2}\right) 1_{\{-\frac{\pi}{2} \leq \lambda \leq \frac{\pi}{2}\}} + 1_{\{\frac{\pi}{2} < \lambda\}}.$$

Notemos que así definidas las funciones de activación son medibles, ya que la imagen inversa de un abierto de $[0, 1]$ siempre será un conjunto medible de \mathbb{R} (capítulo 7 página 77 [Hal74]).

Cabe destacar que la definición tomada es la propuesta en [HSW89] y que existen otras posibles definiciones menos restrictivas con las que también se ha probado la convergencia universal. Por ejemplo podrían aceptarse funciones de activación no continuas (véase [Fun89]); o como se demuestra en [SM15] y en [CC95], funciones de activación no polinómicas y no acotadas.

Para cualquier natural d mayor que cero denotaremos por $\mathcal{A}(\mathbb{R}^d)$ al conjunto de todas las **funciones afines** de \mathbb{R}^d a \mathbb{R} . Es decir el conjunto de funciones de la forma $A(x) = w \cdot x + b$ donde x y w son vectores de \mathbb{R}^d , $b \in \mathbb{R}$ es un escalar y \cdot representa el producto escalar usual. En este contexto, x corresponde al vector entrada de la red neuronal, w los pesos de la red que se multiplicarán con x en la capa intermedia y b el sesgo.

Definición 4.3 (Formalización de una red neuronal de una capa oculta y salida real). Para cualquier función Borel medible G , definida de \mathbb{R} a \mathbb{R} y cualquier natural positivo $d \in \mathbb{N}$ se define a la clase de funciones $\mathcal{H}_G(\mathbb{R}^d, \mathbb{R})$ como

$$\mathcal{H}_G(\mathbb{R}^d, \mathbb{R}) = \{f : \mathbb{R}^d \rightarrow \mathbb{R} / f(x) = \sum_{j=1}^q (\beta_j G(A_j(x))), \\ x \in \mathbb{R}^d, \beta_j \in \mathbb{R}, A_j \in \mathcal{A}(\mathbb{R}^d), q \in \mathbb{N}\}.$$

Conforme avancen los resultado teórico veremos que $\mathcal{H}_G(\mathbb{R}^d, \mathbb{R})$ no depende de la función G seleccionada, así pues tras enunciar tales resultados nos referiremos sin ambigüedad tal conjunto como $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$.

Definiremos a continuación una familia de funciones más generales que $\mathcal{H}_G(\mathbb{R}^d, \mathbb{R})$ con la intención de que actúe como nexo de unión entre la clase de funciones continuas y las redes neuronales de una capa facilitándonos con ello la prueba de los resultados. La familia que introduciremos solo tiene una utilidad teórica, es decir no tendrá ninguna relevancia a nivel práctico en cuanto a implementaciones.

Definición 4.4 (Anillo de aproximación de redes neuronales).

$$\sum \prod^d(G) = \{f : \mathbb{R}^r \rightarrow \mathbb{R} / f(x) = \sum_{j=1}^q \beta_j \prod_{k=1}^{l_j} G(A_{jk}(x)), \\ x \in \mathbb{R}^d, \beta_j \in \mathbb{R}, A_{jk} \in \mathcal{A}(\mathbb{R}^d); l_j, q \in \mathbb{N}\}.$$

Notemos que $\mathcal{H}_G(\mathbb{R}^d, \mathbb{R})$ se recupera en el caso particular en el que $l_j = 1$ para todo j . Los elementos de $\sum \prod^d(G)$ son combinaciones lineales de productos finitos de neuronas.

⊙ **Idea tras la definición de $\mathcal{H}_G(\mathbb{R}^d, \mathbb{R})$.** Nótese que de acorde a nuestra definición 4.1 lo que se ha definido es la clase de las redes neuronales de una capa oculta y salida de una dimensión. Donde cada sumando representa una neurona de la capa oculta.

⊙ **Motivación de la definición de $\sum \prod^d(G)$.** En un principio será más fácil demostrar que con funciones de esta clase seremos capaz de aproximar cualquier función continua. De esta manera este conjunto actuará de nexo de unión entre las funciones continuas y las redes neuronales facilitando las demostraciones. De ahora en adelante nos referiremos a este conjunto como al **de anillo de aproximación** (como curiosidad, el nombre proviene a que tiene estructura de anillo y que se utilizará para aproximar funciones continuas).

4. Las redes neuronales son aproximadores universales

② **Aplicación práctica aprendizaje automático y relación con las funciones medible.** A nivel práctico se tiene un conjunto de datos para los cuales queremos extraer un patrón que nos permita predecir la naturaleza de datos nuevos. Es por ello necesario suponer que estos datos están regidos por alguna regla, la cual puede ser todo lo extraña posible pero que toma valores que pueden ser observables y cuantificables en la mayoría de los casos, estos comportamientos son formalizados matemáticamente con **funciones medibles**.

Introducimos a continuación la notación de los conjuntos de funciones que seremos capaces de aproximar.

Denotamos por $\mathcal{C}(\mathbb{R}^d)$ al conjunto de funciones continuas con dominio en \mathbb{R}^d y codominio \mathbb{R} , por $\mathcal{M}(\mathbb{R}^d)$ al conjunto de todas las funciones Borel medibles de \mathbb{R}^d a \mathbb{R} ; y por B^d a la σ -álgebra de Borel en \mathbb{R}^d .

En lo que respecta a definiciones anteriores, $\mathcal{H}_G(\mathbb{R}^d, \mathbb{R})$ y $\Sigma \Pi^d(G)$ están contenidos en $\mathcal{M}(\mathbb{R}^d)$ para cualquier función Borel-medible G . Si G es continua entonces $\mathcal{H}_G(\mathbb{R}^d, \mathbb{R})$ y $\Sigma \Pi^d(G)$ pertenecen a $\mathcal{C}(\mathbb{R}^d)$. Tengamos presente que $\mathcal{C}(\mathbb{R}^d)$ es un subconjunto de $\mathcal{M}(\mathbb{R}^d)$.

De ahora en adelante nos referiremos a Borel-medible como medible.

4.3.1. Reflexión sobre el tipo de funciones que se pueden aproximar

La existencia de funciones no medibles manifiesta una limitación de la formalización actual de las redes neuronales que plantea las siguientes preguntas:

1. ¿Supone la existencia de este tipo de funciones una verdadera limitación a nivel práctico?
2. ¿Se podría construir alguna arquitectura que sí que las aproximara?

Continuando con el hilo de la segunda cuestión, si se carece de un espacio vectorial, de una medida, ¿Cómo se podría construir una sucesión de funciones que se aproxime? Quizás habría que buscar características más intrínsecas del problema en cuestión, razonamientos topológicos.

Definición 4.5 (Subconjunto denso). Dado un subconjunto S de un espacio métrico (X, ρ) , se dice que S es denso por la distancia ρ en subconjunto T si para todo ϵ positivo y cualquier $t \in T$ existe un $s \in S$ tal que $\rho(s, t) \leq \epsilon$.

Un ejemplo habitual es en el espacio métrico $(\mathbb{R}, |\cdot|)$ con $|\cdot|$ el valor absoluto, el subconjunto $T = \mathbb{R}$ y S los números irracionales, $S = \mathbb{R} \setminus \mathbb{Q}$.

Definición 4.6. Un subconjunto S de $\mathcal{C}(\mathbb{R}^d)$ se dice que es **uniformemente denso para compactos** en $\mathcal{C}(\mathbb{R}^d)$ si para cada subconjunto compacto $K \subset \mathbb{R}^d$ se tiene que S_K es denso según ρ_K en $\mathcal{C}(\mathbb{R}^d)$ donde ρ_K está definida como sigue. Para cualquier $f, g \in \mathcal{C}(\mathbb{R}^d)$

$$\rho_K(f, g) = \sup_{x \in K} |f(x) - g(x)|.$$

Definición 4.7. Una serie de funciones $\{f_n\}$ **converge uniformemente a una función f sobre compactos** si para cada conjunto compacto $K \subset \mathbb{R}^d$ se cumple que

$$\rho_K(f_n, f) \longrightarrow 0 \text{ cuando } n \longrightarrow \infty.$$

② **Noción intuitiva de compacto** Al trabajar con números reales, un espacio es compacto si es cerrado y acotado, lo que a nivel práctico significa que los **datos de entrada se encuentran dentro de un rango concreto**.

② **Noción intuitiva de uniformemente denso para compactos** lo que indica es que **controlamos cuánto de cerca están dos funciones sea cual sea cualquier punto del compacto en que evaluemos** es decir, podríamos afirmar que para una red neuronal que tome valores por ejemplo en $[0, 1]^d$, se puede saber que su error es menor que $\epsilon \in \mathbb{R}^+$ independientemente de la entrada.

② **Idea intuitiva conjunto denso.** Si S es denso en T , se está diciendo que los **elementos de S son capaces de aproximar cualquier elemento de T con la precisión que se desee**.

4.4. Primeros resultados

Teorema 4.2 (Teorema de convergencia real en compactos). *Sea G cualquier función continua no constante definida de \mathbb{R} en \mathbb{R} . Se tiene que $\Sigma\Pi^d(G)$ es uniformemente denso para compactos en $\mathcal{C}(\mathbb{R}^d)$.*

Demostración. Bastará probar que el conjunto $\Sigma\Pi^d(G)$ satisface las hipótesis del teorema de Stone-Weierstrass 3.2. Lo primero será comprobar que $\Sigma\Pi^d(G)$ es un álgebra, para ello veamos que:

1. La función constante uno pertenece al conjunto. Como G no es constante existirá un valor de la imagen distinto de 0, supongamos que $G(a) = b \neq 0$ para $a, b \in \mathbb{R}$. Consideremos la función afín $A(x) = 0 \cdot x + a$, está claro que $\frac{1}{b}G(A(x))$ es la función constantemente uno.
2. El conjunto $\Sigma\Pi^d(G)$ es cerrado para sumas y producto por escalares reales. En efecto, si f, g pertenecen a $\Sigma\Pi^d(G)$, serán de la forma $f = \sum_{j=1}^q \beta_{fj} \prod_{k=1}^{l_{fj}} G(A_{fjk}(x))$ y $g = \sum_{j=1}^p \beta_{gj} \prod_{k=1}^{l_{gj}} G(A_{gjk}(x))$ por lo que

$$\begin{aligned} \gamma f + \sigma g &= \gamma \sum_{j=1}^q \beta_{fj} \prod_{k=1}^{l_{fj}} G(A_{fjk}(x)) + \sigma \sum_{j=1}^p \beta_{gj} \prod_{k=1}^{l_{gj}} G(A_{gjk}(x)) \\ &= \sum_{j=1}^q (\gamma \beta_{fj}) \prod_{k=1}^{l_{fj}} G(A_{fjk}(x)) + \sum_{j=1}^p (\sigma \beta_{gj}) \prod_{k=1}^{l_{gj}} G(A_{gjk}(x)). \end{aligned}$$

Basta reenumerar una de las sumatorias para ver $\gamma f + \sigma g$ como una combinación lineal de productos finitos y por tanto $\gamma f + \sigma g \in \Sigma\Pi^d(G)$.

3. Cerrado para producto. Para $f, g \in \Sigma\Pi^d(G)$, se tiene que fg pertenece a $\Sigma\Pi^d(G)$, para ello basta ver que:

$$\begin{aligned} fg &= \left(\sum_{i \in I_f} \beta_{fi} \prod_{k=1}^{l_i} G(A_{fik}(x)) \right) \left(\sum_{j \in I_g} \beta_{gj} \prod_{k=1}^{l_j} G(A_{gjk}(x)) \right) \\ &= \sum_{i \in I_f} \left(\beta_{fi} \prod_{k=1}^{l_i} G(A_{fik}(x)) \right) \left(\sum_{j \in I_g} \beta_{gj} \prod_{k=1}^{l_j} G(A_{gjk}(x)) \right) \\ &= \sum_{\substack{i \in I_f \\ j \in I_g}} (\beta_{fi} \beta_{gj}) \sum_{k=1}^{l_i+l_j} G(\tilde{A}_{ijk}(x)). \end{aligned}$$

Definimos $\tilde{A}_{ijk}(x) = A_{fik}$ si $k \in \{1, \dots, l_i\}$ y $\tilde{A}_{ijk}(x) = A_{g_{i(k-l_i)}}$ si $k \in \{l_i + 1, \dots, l_i + l_j\}$. luego $fg \in \Sigma\Pi^d(G)$.

Veamos que $\Sigma\Pi^d(G)$ separa puntos para cada compacto $K \subset \mathbb{R}^d$.

Por ser G no constante existirán $a, b \in \mathbb{R}$ distintos cumpliendo que $G(a) \neq G(b)$. Fijadas $x, y \in K$ tomamos entonces cualquiera de las funciones afines que cumplen que $A(x) = a$

Ⓢ **Qué se está probando en el teorema 4.2** Podemos aproximar tanto como queramos cualquier función continua con elementos de $\Sigma\Pi^d(G)$, es decir a lo que llamamos *anillo de aproximación*.

Clave de la demostración del teorema 4.2 Se comprueba primero que para operando elementos del *anillo de aproximación* con sumas, productos y re-escalados no se salen del conjunto, es decir que el resultado es otro elemento del *anillo de aproximación*. Con estas operaciones se irá construyendo y refinando una función que aproxime cualquier función continua con un método cuya idea es con este elemento del anillo de aproximación tengo tal error porque me falla en tales puntos, si le sumo esto otro corrijo ese error y además el resultado sigue perteneciendo al conjunto.

4. Las redes neuronales son aproximadores universales

y $A(y) = b^{-1}$, por lo que $G(A(x)) \neq G(A(y))$ y tenemos como buscábamos que $\Sigma\Pi^d(G)$ separa los puntos de K .

Comprobemos finalmente que para todo punto de K existe una función de $\Sigma\Pi^d(G)$ en el que la imagen no es nula.

Por ser G no constante volvemos a tomar un $a \in \mathbb{R}$ tal que $G(a) \neq 0$, consideramos ahora la aplicación lineal $A(x) = 0 \cdot x + a$ por lo que para todo $x \in K$, $G(A(x)) = G(a) \neq 0$.

Como hemos comprobado se verifican todas las hipótesis del teorema de Stone-Weierstrass, con lo que concluimos que $\Sigma\Pi^d(G)|_K$ es denso en $C(K)$. \square

4.4.1. Observaciones y reflexiones sobre el teorema de convergencia real en compactos

Con esto lo que acabamos de probar que una estructura más general *feedforward neural networks* con tan solo una capa oculta son capaces de aproximar cualquier función continua en un compacto. Cabe destacar que a la función G , que haría el papel de función de activación, solo se le ha pedido como hipótesis ser continua.

Notemos que la función de activación G es única en toda la estructura, sin embargo es habitual la combinación de éstas en una misma red neuronal ([NIGM18], [WLH⁺17], [SVI⁺15]).

Q Nueva hipótesis de optimización El corolario 4.1 abre la puerta a preguntarse si la combinación de diferentes funciones de activación podría mejorar los resultados de alguna manera.

Aportación original

Corolario 4.1 (Pueden combinarse distintas funciones de activación en una misma red neuronal). *Una misma red neuronal puede estar constituida por una familia de funciones continuas no constantes Γ , bastará con generalizar $\Sigma\Pi^d(G)$ a $\Sigma\Pi^d(\Gamma)$ donde*

$$\Sigma\Pi^d(\Gamma) = \{f : \mathbb{R}^d \rightarrow \mathbb{R} / f(x) = \sum_{j=1}^q \beta_j \prod_{k=1}^{l_j} G(A_{jk}(x)), \\ x \in \mathbb{R}^d, \beta_j \in \mathbb{R}, A_{jk} \in \mathcal{A}(\mathbb{R}^d), l_j, q \in \mathbb{N}, G \in \Gamma\}$$

Demostración. La demostración es idéntica a la dada en el Teorema de convergencia real en compactos 4.2. \square

Notemos que este resultado no da pista alguna de las ventajas de una función frente a otra, ni cómo afecta a la *velocidad de convergencia*.

Recordemos que de manera general se ha definido A como una función afín $A(x) = w \cdot x + b$ donde x y w son vectores de \mathbb{R}^d y $b \in \mathbb{R}$ es un escalar. ¿Pero que ocurriría si trabajáramos con transformaciones más generales? Por ejemplo $B((x_1, \dots, x_d)) = \sum_{i=0}^N \sum_{j=0}^d \alpha_{ij} x_j^i$ con N natural positivo.

Q Nueva hipótesis de optimización Gracias al corolario 4.2 nos podemos plantear si aumentar el espacio de búsqueda a partir de transformaciones no lineales es más conveniente que hacerlo mediante el número de neuronas. Abordaremos esta cuestión en ??.

Aportación original

Corolario 4.2 (Generalización de de las transformaciones afines). *Se puede extender $\mathcal{A}(\mathbb{R}^d)$ a conjuntos más generales como el de los polinomios de d variables de grado N , \mathbb{P} .*

¹Sabemos que al menos una habrá, ya que podemos plantear la función afín como un sistema de ecuaciones lineales de $r + 1$ incógnita y 2 soluciones

Demostración. Simplemente hay que reparar en que $\mathcal{A}(\mathbb{R}^d)$ está contenido en el espacio \mathbb{P} . Es más observando la demostración bastará con utilizar cualquier conjunto que contenga a $\mathcal{A}(\mathbb{R}^d)$. \square

Definición 4.8 (Equivalencia entre funciones). Sea μ una medida de probabilidad en $(\mathbb{R}^d, \mathcal{B}^d)$. Dos funciones f y g pertenecientes a $\mathcal{M}(\mathbb{R}^d)$, diremos que son μ -equivalentes si $\mu\{x \in \mathbb{R}^d : f(x) = g(x)\} = 1$.

Lo que se está diciendo es que serán iguales casi por doquier.

Definición 4.9 (Introducción de una distancia basada en una probabilidad). Dada una medida de probabilidad μ en $(\mathbb{R}^d, \mathcal{B}^d)$, se define la métrica ρ_μ definida como

$$\rho_\mu : \mathcal{M}(\mathbb{R}^d) \times \mathcal{M}(\mathbb{R}^d) \longrightarrow \mathbb{R}^+$$

$$\rho_\mu(f, g) = \inf\{\epsilon > 0 : \mu\{x : |f(x) - g(x)| > \epsilon\} < \epsilon\}.$$

Con esta definición lo que se está buscando es una forma de decir cuánto distan las funciones f, g entre ellas.

Lema 4.1 (Caracterización de la convergencia de una sucesión). *Son equivalentes las siguientes afirmaciones:*

1. $\rho_\mu(f_n, f) \longrightarrow 0$.
2. Para cualquier $\epsilon > 0$ se tiene que $\mu\{x : |f_n(x) - f(x)| > \epsilon\} \longrightarrow 0$.
3. $\int \min\{|f_n(x) - f(x)|, 1\} d\mu(x) \longrightarrow 0$.

Demostración. Comenzaremos probando (1) \Rightarrow (2).

Si $\rho_\mu(f_n, f) \longrightarrow 0$ Fijamos $\epsilon_0 > 0$, tenemos por definición que para cualquier $0 < \delta < \epsilon_0$ existirá $n_0 \in \mathbb{N}$ tal que $\rho_\mu(f_n, f) < \delta$ para cada n un natural mayor que n_0 . Es decir,

$$\inf\{\epsilon > 0 : \mu\{x : |f_n(x) - f(x)| > \epsilon\} < \epsilon\} < \delta \quad \forall n \geq n_0$$

entonces

$$\mu\{x : |f_n(x) - f(x)| > \epsilon_0\} \leq \mu\{x : |f_n(x) - f(x)| > \delta\} < \delta \quad \forall n \geq n_0$$

lo que significa que

$$\mu\{x : |f_n(x) - f(x)| > \epsilon_0\} \longrightarrow 0$$

probando con ello la implicación buscada.

Veamos ahora que (2) \Rightarrow (1). Fijamos $\epsilon_0 > 0$ y bajo la hipótesis segunda se tiene que

$$\mu\{x : |f_n(x) - f(x)| > \epsilon_0\} \longrightarrow 0,$$

es decir, que para cualquier real δ cumpliendo que $0 < \delta < \epsilon_0$ existe un natural n_0 a partir del cual todo natural n mayor o igual satisface que

$$\mu\{x : |f_n(x) - f(x)| > \epsilon_0\} \leq \mu\{x : |f_n(x) - f(x)| > \delta\} < \delta \quad \forall n \geq n_0$$

lo que significa que

② **Idea intuitiva equivalencia de funciones:**

En la definición 4.8 Se considera que dos funciones son equivalentes si son iguales en casi todos sus puntos. El objetivo de todas estas definiciones es **saber cuándo una red neuronal es equivalente a una función medible o cuánto la aproxima.**

② **Idea intuitiva distancia probabilidad:**

Este concepto de analizar la tendencia de la mayoría de los puntos se ve reflejado en la definición de distancia basada en una probabilidad 4.9, siendo entonces la **distancia de dos funciones la menor de las distancias que siguen la mayoría de sus puntos.**

El lema 4.1 nos permitirá trabajar con mayor comodidad matemática este concepto.

4. Las redes neuronales son aproximadores universales

$$\inf\{\epsilon > 0 : \mu\{x : |f_n(x) - f(x)| > \epsilon\} < \epsilon\} < \delta \quad \forall n \geq n_0$$

que por definición de la distancia equivale a que

$$\rho_\mu(f_n, f) < \delta \quad \forall n \geq n_0$$

probando con ello

$$\rho_\mu(f_n, f) \longrightarrow 0.$$

Probaremos ahora que (2) \implies (3).

Por (2) se tiene que sea cual sea el ϵ cumpliendo que $0 < \epsilon \leq 2$ existirá un natural n_0 a partir del cual, cualquier otro natural n satisface que

$$\mu\{x : |f_n(x) - f(x)| > \frac{\epsilon}{2}\} < \frac{\epsilon}{2},$$

Gracias a esta desigualdad, para cualquier $n > n_0$ podemos acotar la siguiente integral:

$$\int \min\{|f_n(x) - f(x)|, 1\} d\mu(x) \leq \frac{\epsilon}{2} \left(1 - \frac{\epsilon}{2}\right) + 1 \frac{\epsilon}{2} = \epsilon - \frac{\epsilon^2}{4} < \epsilon.$$

probando con ello la implicación (2) \implies (3).

Finalmente comprobaremos la implicación (3) \implies (1).

Para cada $n \in \mathbb{N}$ llamamos $g_n = \min\{|f_n - f|, 1\}$. Por (2), dado $0 < \epsilon < 1$, existe un $n_0 \in \mathbb{N}$ de modo que si $n \geq n_0$ se cumple que

$$\int g_n d\mu < \epsilon^2 \tag{4.1}$$

Como $\epsilon < 1$ tenemos que

$$\{x; g_n(x) > \epsilon\} = \{x; |f_n - f| > \epsilon\}$$

luego

$$\mu\{x; |f_n - f(x)| > \epsilon\} = \mu\{x; g_n(x) > \epsilon\} \leq \frac{1}{\epsilon} \int_{g_n(x) > \epsilon} g_n d\mu < \epsilon \quad \forall n \geq n_0$$

donde se ha usado la desigualdad de Chebyshev para g_n y la desigualdad (4.1).

Probando con esto lo buscado que para cualquier $\epsilon > 0$ se tiene que

$$\mu\{x : |f_n(x) - f(x)| > \epsilon\} \longrightarrow 0.$$

□

Lema 4.2. Si $\{f_n\}$ es una sucesión de funciones en $\mathcal{M}(\mathbb{R}^d)$ que converge uniformemente en un compacto a f entonces $\rho_\mu(f_n, f) \longrightarrow 0$.

Demostración. Para cada $n \in \mathbb{N}$ llamamos $g_n = \min\{|f_n - f|, 1\}$. Tengamos presente que por el lema 4.1 deberemos probar que para cualquier $\epsilon > 0$, existe un n_0 natural, tal que para cualquier otro natural n mayor o igual que n_0 se tiene que

$$\int \min\{|f_n(x) - f(x)|, 1\} d\mu(x) < \frac{\epsilon}{2}.$$

Sea $\mu(\mathbb{R}^d) = M \in \mathbb{R}^+$ y sin pérdida de generalidad puede suponerse $M = 1$ ². Ya que \mathbb{R}^r es un espacio métrico localmente compacto (pag 228 teorema 52.G [Hal74]), se tiene que existirá un subconjunto K compacto de \mathbb{R}^r con medida $\mu(K) > 1 - \frac{\epsilon}{2}$. Para el cual, por su compacidad, existirá un n_0 natural $\sup_{x \in K} |f_n(x) - f(x)| < \frac{\epsilon}{2}$ para cada natural n con $n \geq n_0$.

De modo que para cualquier $x \in K$, n con $n \geq n_0$ se cumple que

$$|f_n(x) - f(x)| = \min\{|f_n(x) - f(x)|, 1\} = g_n.$$

Por lo que

$$\int_K g_n d\mu \leq \mu(K) \sup_{x \in K} |f_n(x) - f(x)| \leq \frac{\epsilon}{2}. \quad (4.2)$$

Acotando el primer sumando por la medida del complemento de la región integrada y en virtud de (4.2)

$$\begin{aligned} \int_{\mathbb{R}^d \setminus K} \min\{|f_n(x) - f(x)|, 1\} d\mu(x) + \int_K \min\{|f_n(x) - f(x)|, 1\} d\mu(x) \\ \leq \mu(\mathbb{R}^d \setminus K) + \frac{\epsilon}{2} \leq \frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon \end{aligned}$$

para cualquier $n \geq n_0$. □

Lema 4.3. Para cualquier medida finita μ se tiene que $\mathcal{C}(\mathbb{R}^d)$ es denso en $\mathcal{M}(\mathbb{R}^d)$ para la distancia ρ_μ .

Demostración. Dada cualquier $f \in \mathcal{M}(\mathbb{R}^d)$ y un $\epsilon > 0$ arbitrario, tenemos que encontrar una función g que cumpla que $\rho_\mu(f, g) < \epsilon$.

Tomando un $M > 1$ lo suficientemente grande, tenemos que

$$\int \min\{|f(x) 1_{|f(x)| < M} - f(x)|, 1\} d\mu(x) < \frac{\epsilon}{2}.$$

Sabemos además que podemos aproximar $f 1_{|f| < M}$ por g , una función continua que es límite de una sucesión de funciones simples (pag 241-242, teoremas 55C y 55D [Hal74]), la cual satisface

$$\int \min\{|f(x) 1_{|f(x)| < M} - g(x)|, 1\} d\mu(x) < \frac{\epsilon}{2}. \quad (4.3)$$

Tomamos M lo suficientemente grande, de tal forma que

$$\mu(\{x : |f(x)| \geq M\}) < \frac{\epsilon}{2} \quad (4.4)$$

y denotamos por Λ al conjunto $\{x : |f(x)| < M\}$.

Concluyendo por 4.3 y 4.4

$$\begin{aligned} \int \min\{|f - g|, 1\} d\mu &= \int_\Lambda \min\{|f 1_{|f(x)| < M} - g|, 1\} d\mu + \int_{\mathbb{R}^d \setminus \Lambda} \min\{|f - g|, 1\} d\mu \\ &< \frac{\epsilon}{2} + \mu(\{x : |f(x)| \geq M\}) < \frac{\epsilon}{2} + \frac{\epsilon}{2} < \epsilon. \end{aligned}$$

□

Teorema 4.3. Para cualquier función continua no constante G , $d \in \mathbb{N}$ y medida de probabilidad μ en $(\mathbb{R}^d, \mathcal{B}^d)$, se tiene que $\sum \Pi^d(G)$ es ρ_μ -denso en $\mathcal{M}(\mathbb{R}^d)$.

⊙ **Idea intuitiva lema 4.3:**

Las funciones continuas pueden tomar formas muy variopintas, estando incluso no acotadas. La función de Dirichlet definida en $D(x) = 1$ si x es irracional y $D(x) = 0$ si x es racional, es medible pero no es continua ya que presenta infinitas discontinuidades.

Sin embargo, las funciones continuas son más simples, fáciles de entender y manejar. Gracias al lema 4.3 acabamos de probar que **podemos aproximar en casi todos sus puntos cualquier función medible a partir de una continua.**

Por ejemplo la función constantemente uno aproxima a la función D previamente definida.

4. Las redes neuronales son aproximadores universales

Demostración. Debemos probar que para cualquier función $f \in \mathcal{M}(\mathbb{R}^d)$ existe una sucesión de funciones $\{h_n\}_{n \in \mathbb{N}}$ contenida en $\Sigma \Pi^d(G)$ y cumpliendo que $\rho_\mu(h_n, f) \rightarrow 0$.

Consideramos cualquier $f \in \mathcal{M}(\mathbb{R}^d)$, por el lema 4.3 sabemos que $\mathcal{C}(\mathbb{R}^d)$ es ρ_μ -denso en $\mathcal{M}(\mathbb{R}^d)$; es decir, existirá una sucesión $\{f_n\}_{n \in \mathbb{N}}$ de funciones de $\mathcal{C}(\mathbb{R}^d)$ convergente a f .

Por otra parte sabemos por el teorema 4.2, que $\Sigma \Pi^d(G)$ es uniformemente denso por compactos en $\mathcal{C}(\mathbb{R}^d)$, luego en cualquier compacto $K \subset \mathbb{R}^d$ existirá una sucesión (con n fijo) $\{g(n)_m\}_{m \in \mathbb{N}}$ convergente a f_n , el término n -ésimo de la sucesión convergente a g .

Así pues, denotando como h_n al término $g(n)_n$, obtenemos una sucesión de funciones en $\mathcal{M}(\mathbb{R}^d)$ que converge uniformemente en compactos a f y por el lema 4.2 tenemos que $\rho_\mu(h_n, f) \rightarrow 0$ como queríamos probar. \square

⊗ Idea intuitiva lema

4.4

Hasta ahora hemos probado que $\Sigma \Pi^d(G)$ es capaz de aproximar funciones medibles pero siendo G una función de activación continua, vamos a relajar tal hipótesis a que G sea cualquier función de activación, sin exigirle continuidad.

Lema 4.4. Sea F una función de activación continua y ψ una función de activación arbitraria. Para cualquier $\epsilon > 0$ existe un elemento H_ϵ de $\mathcal{H}_\psi(\mathbb{R}, \mathbb{R})$ cumpliendo que

$$\sup_{\lambda \in \mathbb{R}} |F(\lambda) - H_\epsilon(\lambda)| < \epsilon.$$

Demostración. Procedamos a realizar la siguiente prueba constructiva. Tomamos fijo pero arbitrario un $\epsilon > 0$, que sin pérdida de generalidad supondremos menor que uno. Para que H_ϵ pertenezca a $\mathcal{H}_\psi(\mathbb{R}, \mathbb{R})$ deberá de ser de la forma $\sum_{j=1}^{q-1} b_j \psi(A_j(\lambda))$ debemos encontrar por ende el número de sumatorias, $q - 1$; esa misma cantidad de constantes reales b_j y funciones afines A_j . Para ello tomamos como q cualquier número natural que cumpla que

$$\frac{1}{q} < \frac{\epsilon}{4}. \tag{4.5}$$

Fijaremos para cada $j \in \{1, 2, \dots, q - 1\}$ los coeficientes b_j como $\frac{1}{q}$.

Seleccionamos cualquier constante real $M > 0$ de tal forma que se cumpla que

$$\psi(-M) < \frac{\epsilon}{2q} \quad \text{y} \quad \psi(M) > 1 - \frac{\epsilon}{2q}. \tag{4.6}$$

Sabemos que esto es posible ya que por ser ψ una función de activación satisface que $\lim_{\lambda \rightarrow \infty} \psi(\lambda) = 1$ y que $\lim_{\lambda \rightarrow -\infty} \psi(\lambda) = 0$, por tanto existirá una constante M_1 positiva tal que a partir de ella cualquier otra constante n_1 mayor o igual que cumpla que $\psi(n_1) > 1 - \frac{\epsilon}{2q}$. También existirá una constante M_2 positiva tal que a partir de ella cualquier otra constante n_2 mayor o igual tal que $\psi(-n_2) < \frac{\epsilon}{2q}$. Podemos tomar como M al máximo de M_1 y M_2 .

Seleccionaremos ahora los siguientes puntos del dominio

$$\begin{aligned} r_j &= \sup \left\{ \lambda : F(\lambda) = \frac{j}{q} \right\}, \quad \text{con } j \in \{1, \dots, q - 1\} \\ r_q &= \sup \left\{ \lambda : F(\lambda) = 1 - \frac{1}{2q} \right\}. \end{aligned} \tag{4.7}$$

Que por ser F una función de activación continua sabemos que existen ya que $\lim_{\lambda \rightarrow -\infty} F(\lambda) = 0$ y $\lim_{\lambda \rightarrow \infty} F(\lambda) = 1$.

²De otra forma bastaría con definir en los pasos siguientes $\mu(K) > M - \frac{\epsilon}{2}$ y acotar con $\frac{\epsilon}{2M}$ en vez de $\frac{\epsilon}{2}$.

⊗ Idea intuitiva teorema 4.3

Se prueba en el teorema 4.3 que con una versión más general de una red neuronal (perteneciente a $\Sigma \Pi^d(G)$) se es capaz de aproximar cualquier función medible. La idea de la demostración es sencilla, sabemos aproximar una función medible con una continua y a su vez una continua con una red neuronal generalizada, luego sabemos aproximar una función medible con una red neuronal generalizada.

Procedemos ahora a definir las distintas aplicaciones afines. Para cualesquiera reales s, r que cumplan $r < s$ sea $A_{rs} \in \mathcal{A}(\mathbb{R})$ la única aplicación afín que satisface que

$$A_{rs}(r) = -M \text{ y } A_{rs}(s) = M.$$

Acabamos pues de determinar todos los elementos que conforman a H_ϵ , de tal forma que se tiene que

$$H_\epsilon(\lambda) = \frac{1}{q} \sum_{j=1}^{q-1} \psi(A_{r_j, r_{j+1}}(\lambda))$$

y así definida cumple que:

- Si $\lambda \in (-\infty, r_1]$: Se cumple que $\lambda \leq r_1 < r_2 < \dots < r_q$ luego para todos $j \in \{1, \dots, q-1\}$ las funciones afines satisfacen que $A_{r_j, r_{j+1}}(\lambda) < -M$ y por cómo se fijó M en la condición 4.6 resulta que $\psi(A_{r_j, r_{j+1}}(\lambda)) < \frac{\epsilon}{2q}$. Por tanto para $\lambda \in (-\infty, r_1]$ se tiene

$$H_\epsilon(\lambda) = \frac{1}{q} \sum_{j=1}^{q-1} \psi(A_{r_j, r_{j+1}}(\lambda)) < \frac{1}{q} \sum_{j=1}^{q-1} \frac{\epsilon}{2q} < \frac{1}{q} (q-1) \frac{\epsilon}{2q} < \frac{\epsilon}{2q} < \frac{\epsilon}{2}$$

y como además $0 \leq F(\lambda) \leq \frac{1}{q} < \frac{\epsilon}{2}$ por cómo se seleccionaron los puntos r_j en 4.7 se tiene que

$$|F(\lambda) - H_\epsilon(\lambda)| < \frac{\epsilon}{2} + \frac{\epsilon}{2} < \epsilon.$$

- Si $\lambda \in (r_q, +\infty)$: Se cumple que $r_1 < r_2 < \dots < r_q < \lambda$ luego para todo $j \in \{1, \dots, q-1\}$ las funciones afines satisfacen que $A_{r_j, r_{j+1}}(\lambda) > M$ y por cómo se fijó M en 4.6 resulta que $\psi(A_{r_j, r_{j+1}}(\lambda)) > 1 - \frac{\epsilon}{2q}$ concluyendo que para $\lambda \in (r_q, +\infty)$:

$$1 \geq H_\epsilon(\lambda) = \frac{1}{q} \sum_{j=1}^{q-1} \psi(A_{r_j, r_{j+1}}(\lambda)) > \frac{1}{q} \sum_{j=1}^{q-1} \left(1 - \frac{\epsilon}{2q}\right) > \frac{(q-1)}{q} \left(1 - \frac{\epsilon}{2q}\right)$$

y por ende, como además $\frac{(q-1)}{q} \left(1 - \frac{\epsilon}{2q}\right) < \frac{q-1}{q} \leq F(\lambda) \leq 1$ por cómo se seleccionaron los puntos r_j en 4.7 se tiene que

$$|F(\lambda) - H_\epsilon(\lambda)| \leq 1 - \frac{(q-1)}{q} \left(1 - \frac{\epsilon}{2q}\right) = \frac{1}{q} + \frac{\epsilon}{2q} < \epsilon.$$

Donde para acotar $\frac{1}{q}$ hemos usado la desigualdad 4.5.

- Si $\lambda \in (r_j, r_{j+1}]$ para $j \in \{1, \dots, q-1\}$:

Tenemos por una parte que $\frac{j}{q} < F(\lambda) \leq \frac{j+1}{q}$ y podemos descomponer H_ϵ en las siguientes sumatorias:

$$qH_\epsilon(\lambda) = \sum_{i=1}^{j-1} \psi(A_{r_i, r_{i+1}}(\lambda)) + \psi(A_{r_j, r_{j+1}}(\lambda)) + \sum_{i=j+1}^{q-1} \psi(A_{r_i, r_{i+1}}(\lambda))$$

Nótese que así definida, si $j = q-2$ o $j = q-1$ el último sumando no aparece.

⊙ **Idea intuitiva demostración del lema 4.4**

Aprovechamos que sabemos que la función de activación continua F es no decreciente así que podemos dividir su imagen en trozos iguales y aproximar la imagen con una función en escalera que valga en cada tramo el valor mínimo. A su vez, una función en escalera se puede construir con una red neuronal de una capa oculta que tenga como función de activación arbitraria ψ , cada vez que se activa un nodo más se sube un escalón más de la escalera.

4. Las redes neuronales son aproximadores universales

Los términos de la primera sumatoria serán mayores que $\left(1 - \frac{\epsilon}{2q}\right)$ y menores o iguales que la unidad, el segundo sumando satisface que $0 \leq q\psi(A_{r_j, r_{j+1}}(\lambda)) \leq 1$ y para la última sumatoria, todos sus términos serán menores que $\frac{\epsilon}{2q}$ y mayores o iguales que cero. De donde resulta que :

$$\frac{j-1}{q} \left(1 - \frac{\epsilon}{2q}\right) < H_\epsilon(\lambda) < \frac{j-1}{q} + \frac{1}{q} + \frac{q-j}{q} \frac{\epsilon}{2q},$$

concluyendo que

$$F(\lambda), H_\epsilon(\lambda) \in \left[\frac{j-1}{q} \left(1 - \frac{\epsilon}{2q}\right), \frac{j+1}{q} \right]$$

y por tanto:

$$|F(\lambda) - H_\epsilon(\lambda)| \leq \frac{j+1}{q} - \frac{j-1}{q} \left(1 - \frac{\epsilon}{2q}\right) = \frac{2}{q} + \frac{j-1}{q} \frac{\epsilon}{2q} < \frac{\epsilon}{2} + \frac{\epsilon}{2} < \epsilon.$$

La acotación $|F(\lambda) - H_\epsilon(\lambda)| < \epsilon$ se cumple para todo

$$\lambda \in (-\infty, r_1] \cup (r_q, +\infty) \cup \bigcup_{j \in \{1, \dots, q-1\}} (r_j, r_{j+1}] = \mathbb{R},$$

probando con ello lo buscado. □

Teorema 4.4. Para cualquier función de activación ψ , d natural positivo y medida de probabilidad μ en (\mathbb{R}^d, B^d) , se tiene que $\Sigma \Pi^d(\psi)$ es uniformemente denso en compactos en $\mathcal{C}(\mathbb{R}^d)$ y denso en $\mathcal{M}(\mathbb{R}^d)$ para la distancia ρ_μ .

④ **Idea intuitiva teorema 4.4**

Con una versión generalizada de una red neuronal que utilice cualquier función de activación (es decir que pertenezca a $\Sigma \Pi^d(\psi)$) **podemos aproximar funciones continuas y funciones medibles.**

Esto ha supuesto relajar la hipótesis de G con respecto al último teorema 4.3 de convergencia a funciones.

Demostración. En virtud del lema 4.2 y del teorema 4.3 basta con probar que $\Sigma \Pi^d(\psi)$ es uniformemente denso en compactos de $\Sigma \Pi^d(F)$, donde F es una función de activación continua ³.

Para ello basta ver que cualquier función de la forma $\prod_{k=1}^l F(A_k(\cdot))$ puede ser uniformemente aproximada por una sucesión de funciones de $\Sigma \Pi^d(\psi)$.

Fijamos $\epsilon > 0$. Por continuidad, existirá un $\delta > 0$ tal que, para cualesquiera números reales $0 \leq a_k, b_k \leq 1$, con $k \in \{1, \dots, l\}$ cumpliendo que $|a_k - b_k| < \delta$ se tiene

$$\left| \prod_{k=1}^l a_k - \prod_{k=1}^l b_k \right| < \epsilon. \quad (4.8)$$

Por el lema 4.4 existe una función $H_\delta(\cdot) = \sum_{t=1}^T \beta_t \psi(A_t(\cdot))$ cumpliendo que

$$\sup_{\lambda \in \mathbb{R}} |F(\lambda) - H_\delta(\lambda)| < \delta.$$

Usando 4.8 para $a_k = F(A_k(x))$ y $b_k = H_\delta(A_k(x))$ obtenemos

$$\sup_{x \in \mathbb{R}^d} \left| \prod_{k=1}^l F(A_k(x)) - \prod_{k=1}^l H_\delta(A_k(x)) \right| < \epsilon. \quad (4.9)$$

³el razonamiento por el que con esta hipótesis es suficiente es idéntico al realizado para la demostración del teorema 4.3.

Puesto que H_δ es de la forma $\sum_{t=1}^T \beta_t \psi(A_t(\cdot))$ y porque $A_t(A_k(\cdot)) \in \mathcal{A}(\mathbb{R}^d)$ se tiene por la desigualdad 4.9 que $\prod_{k=1}^l H_\delta(A_k(\cdot)) \in \Sigma \Pi^d(\psi)$.

Por lo tanto $\prod_{k=1}^l F(A_k(\cdot))$ puede ser aproximado por elementos de $\Sigma \Pi^d(\psi)$ y acabamos de probar con ello lo buscado. \square

Lema 4.5. Para cada función de activación ψ , cada $\epsilon > 0$ y cada $M > 0$ existe una función $\cos_{M,\epsilon} \in \mathcal{H}_F(\mathbb{R}, \mathbb{R})$ tal que

$$\sup_{\lambda \in [-M, M]} |\cos_{M,\epsilon}(\lambda) - \cos(\lambda)| < \epsilon.$$

Demostración. Sea F la función de activación *cosine squasher* de Gallant and White (1988) definida en 4.2 Comenzamos probando que para un intervalo acotado $[-M, M]$, existe $H_{[-M, M]} \in \mathcal{H}_F(\mathbb{R}, \mathbb{R})$ tal que para todo elemento $\lambda \in [-M, M]$ se cumple que

$$H_{[-M, M]}(\lambda) = \cos(\lambda).$$

Calculamos $H_{[-M, M]} \in \mathcal{H}_F(\mathbb{R}, \mathbb{R})$ de forma constructiva. Recordemos que

$$F(\lambda) = \left(1 + \cos\left(\lambda - \frac{\pi}{2}\right) \frac{1}{2}\right) 1_{\{-\frac{\pi}{2} \leq \lambda \leq \frac{\pi}{2}\}} + 1_{\{\frac{\pi}{2} < \lambda\}}.$$

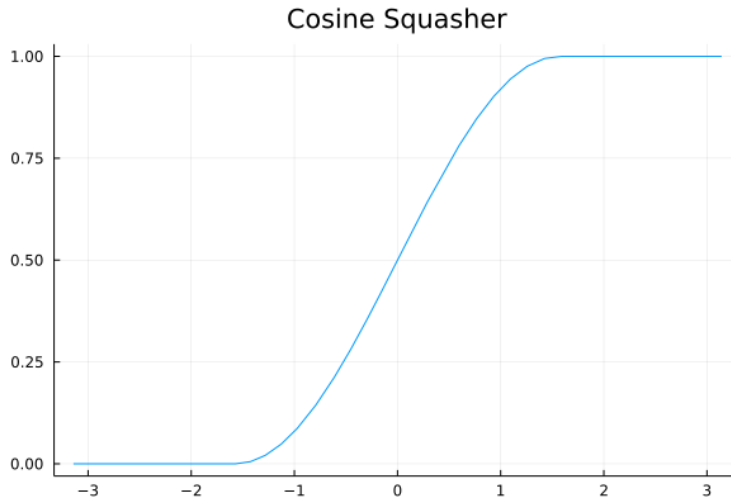


Figura 4.3.: Función *cosine squasher*

Se tiene pues que para cualquier $\lambda \in [-\frac{\pi}{2}, \frac{\pi}{2}]$ se cumple

$$2F(\lambda) - 1 = \cos\left(\lambda - \frac{\pi}{2}\right)$$

y, haciendo $\mu = \lambda - \frac{\pi}{2}$, resulta que para cualquier $\mu \in [-\pi, 0]$ se tiene

$$\cos(\mu) = 2F\left(\mu + \frac{\pi}{2}\right) - 1$$

4. Las redes neuronales son aproximadores universales

Además, puesto que $F(\mu + 2\pi M) = 1$ para todo $\mu \in [-M, M] \supset [-\pi, 0]$, para cada $\mu \in [-\pi, 0]$ podemos escribir

$$\cos(\mu) = 2F\left(\mu + \frac{\pi}{2}\right) - F(\mu + 2\pi M).$$

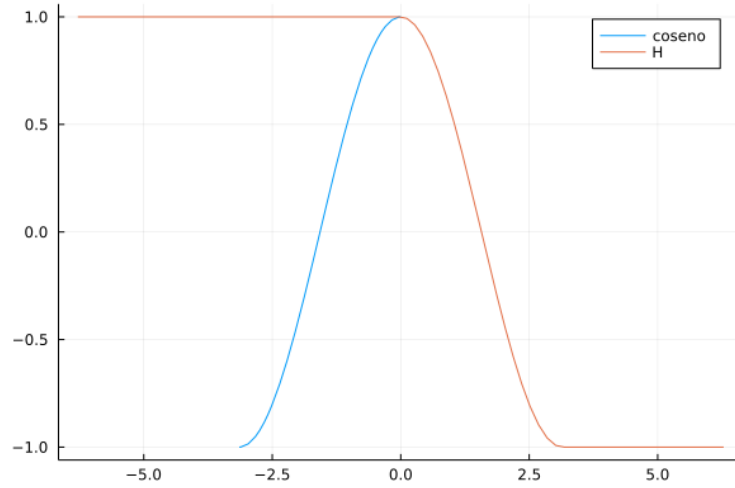


Figura 4.4.: Comparativa $H_{[-\pi,0]}$ con la función coseno real.

Podemos definir entonces

$$H_{[-\pi,0]} = 2F\left(\mu + \frac{\pi}{2}\right) - F(\mu + 2\pi M)$$

que cumple que $H_{[-\pi,0]} = \cos$ en $[-\pi, 0]$

Por la simetría de la función coseno resulta que para todo $\mu \in [0, \pi]$ se tiene

$$\cos(\mu) = \cos(-\mu) = H_{[-\pi,0]}(-\mu).$$

Así pues podemos definir

$$\begin{aligned} H_{[-\pi,\pi]}(\mu) &= H_{[-\pi,0]}(\mu) + H_{[-\pi,0]}(-\mu) - 1 \\ &= H_{[-\pi,0]}(\mu) + H_{[-\pi,0]}(-\mu) - F(\mu + 2\pi M) \end{aligned}$$

que cumple que $H_{[-\pi,\pi]} = \cos$ en $[-\pi, \pi]$. Fijamos un natural N cumpliendo que $2\pi N \geq M$ y definimos

$$\begin{aligned} H_{[-2\pi N, 2\pi N]}(\lambda) &= \sum_{j=1}^N (H_{[-\pi,\pi]}(\lambda + 2\pi j) + 1) + H_{[-\pi,\pi]}(\lambda) \\ &\quad - \sum_{j=1}^N (H_{[-\pi,\pi]}(-\lambda + 2\pi j - \pi) + 1), \end{aligned}$$

$$H_{[-2\pi N, 2\pi N]}(\lambda) = H_{[-\pi,\pi]}(\lambda) + \sum_{j=1}^N (H_{[-\pi,\pi]}(\lambda + 2\pi j) - H_{[-\pi,\pi]}(-\lambda + 2\pi j - \pi)).$$

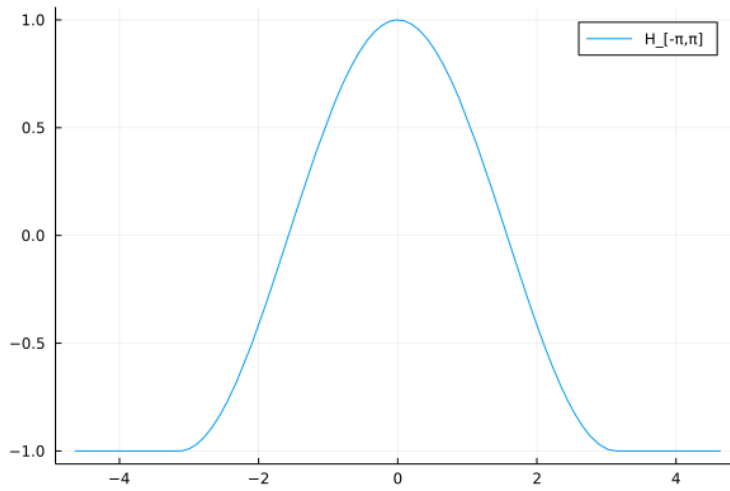
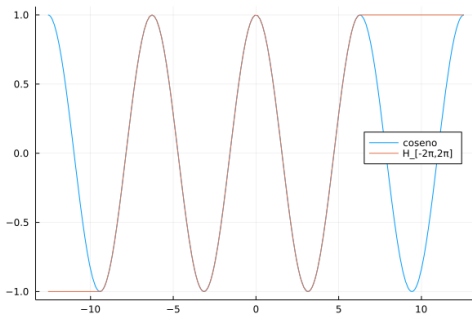
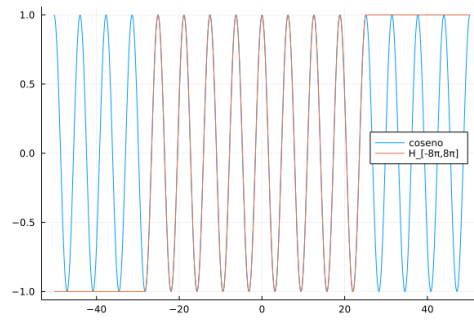


Figura 4.5.: Función $H_{[-\pi, \pi]}$.



(a) Función $H_{[-2\pi, 2\pi]}$ con $N = 1$.



(b) Función $H_{[-8\pi, 8\pi]}$ con $N = 4$.

Figura 4.6.: Ejemplos de funciones $H_{[-M, M]}$

Usando la periodicidad del coseno es fácil ver que $H_{[-2\pi N, 2\pi N]} = \cos$ en $[-2\pi N, 2\pi N]$. Está claro que por ser $2\pi N \geq M$ podemos definir finalmente

$$H_{[-M, M]} = H_{[-2\pi N, 2\pi N]}$$

lo que concluye finalmente la construcción de $H_{[-M, M]}$. La construcción anterior nos dice que $H_{[-M, M]} \in \mathcal{H}_F(\mathbb{R}, \mathbb{R})$. De hecho existen $t \in \mathbb{N}$ y traslaciones A_j para $j \in \{1, \dots, t\}$ de modo que

$$H_{[-M, M]} = \sum_{j=1}^t \alpha_j A_j(\lambda) \quad \alpha_j \in \{-1, 1\}.$$

Como F es una función de activación continua, el lema 4.4 nos da la existencia de $W_{\frac{\epsilon}{t}} \in \mathcal{H}_\psi(\mathbb{R}, \mathbb{R})$ cumpliendo que $|F - W_{\frac{\epsilon}{t}}| < \frac{\epsilon}{t}$ en todo \mathbb{R} . Por tanto

$$|F \circ A_i - W_{\frac{\epsilon}{t}} \circ A_i| < \frac{\epsilon}{t} \quad \forall j \in \{1, \dots, t\}.$$

4. Las redes neuronales son aproximadores universales

Podemos ya definir la función que buscamos:

$$\cos_{M,\epsilon}(\lambda) = \sum_{j=1}^t \alpha_j W_{\frac{\epsilon}{t}}(A_j(\lambda)).$$

De esta manera, para cualquier $\lambda \in [-M, M]$

$$\begin{aligned} |\cos_{M,\epsilon}(\lambda) - \cos(\lambda)| &\leq |\cos(\lambda) - H_{[-M,M]}(\lambda)| + |\cos_{M,\epsilon}(\lambda) - H_{[-M,M]}(\lambda)| \\ &\leq 0 + |\cos_{M,\epsilon}(\lambda) - H_{[-M,M]}(\lambda)| \\ &\leq \sum_{j=1}^t |W_{\frac{\epsilon}{t}}(A_j(\lambda)) - F(A_j(\lambda))| \\ &< \sum_{j=1}^t \frac{\epsilon}{t} = \epsilon. \end{aligned}$$

Dando con esto por concluida la demostración. □

Lema 4.6. Sea $g(\cdot) = \sum_{j=1}^q \beta_j \cos(A_j(\cdot))$ con $A_j \in \mathcal{A}(\mathbb{R}^d)$ y $\beta_j \in \mathbb{R}$. Para cualquier función de activación ψ , cualquier compacto $K \subset \mathbb{R}^d$ y cualquier $\epsilon > 0$ existe $f \in \mathcal{H}_{\psi}(\mathbb{R}^d, \mathbb{R})$ para el cual se cumple que

$$\sup_{x \in K} |g(x) - f(x)| < \epsilon.$$

Demostración. Puesto que K es un compacto, el número de sumatorias q es finito y A_j son funciones continuas, existe M un número real positivo cumpliendo que

$$A_j(K) \subseteq [-M, M] \quad \text{para todo } j \in \{1, \dots, q\}.$$

Definimos

$$B = \max\{|\beta_j| : j \in \{1, \dots, q\}\}$$

En virtud del lema 4.5 podemos encontrar $\cos_{M, \frac{\epsilon}{qB}} \in \Sigma(\psi)$ cumpliendo que

$$\sup_{\lambda \in [-M, M]} |\cos_{M, \frac{\epsilon}{qB}}(\lambda) - \cos(\lambda)| < \frac{\epsilon}{qB}$$

por lo que para cualquier $j \in \{1, \dots, q\}$ se tiene que

$$\sup_{x \in K} |\cos_{M, \frac{\epsilon}{qB}}(A_j(x)) - \cos(A_j(x))| \leq \sup_{\lambda \in [-M, M]} |\cos_{M, \frac{\epsilon}{qB}}(\lambda) - \cos(\lambda)| < \frac{\epsilon}{qB}.$$

Definamos ahora f como

$$f(x) = \sum_{j=1}^q \beta_j \cos_{M, \frac{\epsilon}{qB}}(A_j(x)),$$

que cumple $f \in \mathcal{H}_\psi(\mathbb{R}^d, \mathbb{R})$ puesto que $\cos_{M, \frac{\epsilon}{qB}} \in \mathcal{H}_\psi(\mathbb{R}, \mathbb{R})$, y satisface además que

$$\begin{aligned} \sup_{x \in K} |f(x) - g(x)| &\leq \sup_{\lambda \in [-M, M]} \left| \sum_{j=1}^q \beta_j \cos_{M, \frac{\epsilon}{qB}}(\lambda) - \sum_{j=1}^q \beta_j \cos(\lambda) \right| \\ &\leq \sup_{\lambda \in [-M, M]} \sum_{j=1}^q |\beta_j| |\cos_{M, \frac{\epsilon}{qB}}(\lambda) - \cos(\lambda)| \\ &< \sup_{\lambda \in [-M, M]} \sum_{j=1}^q B \frac{\epsilon}{qB} = \epsilon. \end{aligned}$$

Con lo que acabamos de probar la tesis del lema. □

Lema 4.7. Para cualquier función de activación ψ se tiene que $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ es uniformemente denso para compactos en $\mathcal{C}(\mathbb{R}^d)$.

Demostración. En virtud del lema 4.6 bastaría probar que $\mathcal{H}_{\cos}(\mathbb{R}^d, \mathbb{R})$ es uniformemente denso en compactos, ya que hemos visto que $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ es denso en $\mathcal{H}_{\cos}(\mathbb{R}^d, \mathbb{R})$.

Veamos ahora que $\mathcal{H}_{\cos}(\mathbb{R}^d, \mathbb{R}) = \Sigma \Pi^d(\cos)$ donde recordamos que $\Sigma \Pi^d(\cos)$ se correspondía con el espacio de los polinomios trigonométricos.

$$\Sigma \Pi^d(\cos) = \left\{ \sum_{j=1}^q \beta_j \prod_{k=1}^{l_j} \cos(A_{j,k}(\cdot)) : q, l_j \in \mathbb{N}, \beta_j \in \mathbb{R}, A_{j,k} \in \mathcal{A}(\mathbb{R}^d) \right\},$$

y $\mathcal{H}_{\cos}(\mathbb{R}^d, \mathbb{R})$ el caso particular de los polinomios anteriores de grado uno. Luego está claro que $\mathcal{H}_{\cos}(\mathbb{R}^d, \mathbb{R}) \subseteq \Sigma \Pi^d(\cos)$.

Para cualesquiera números reales A, B se tiene

$$\frac{1}{2}(\cos(A + B) + \cos(A - B)) = \cos(A) \cos(B).$$

y de manera general si definimos

$$\Lambda_l = \{(x_1, x_2, \dots, x_l) : x_1 = 1, x_i \in \{-1, 1\} \text{ para cualquier } 1 < i \leq l\}$$

se cumple (por inducción)

$$\prod_{k=1}^l \cos(A_k) = \frac{1}{2^{l-1}} \left(\sum_{x \in \Lambda_l} \cos \left(\sum_{i=1}^l x_i A_i \right) \right).$$

Por lo que cada elemento $\sum_{j=1}^q \beta_j \prod_{k=1}^{l_j} \cos(A_{j,k}(\cdot)) \in \Sigma \Pi^d(\cos)$ podrá ser expresado como $\sum_{j=1}^q \beta_j \prod_{k=1}^{l_j} \cos(A_{j,k}(\cdot)) = \sum_{j=1}^t \alpha_j \cos(A_{j,k}(\cdot))$, es decir, como un elemento de $\mathcal{H}_{\cos}(\mathbb{R}^d, \mathbb{R})$. Concluimos como queríamos que son iguales los espacios $\mathcal{H}_{\cos}(\mathbb{R}^d, \mathbb{R})$ y $\Sigma \Pi^d(\cos)$.

Finalmente gracias al teorema 4.2, por ser el coseno una función real continua no constante sabemos que $\Sigma \Pi^d(\cos)$ es uniformemente denso en compactos en $\mathcal{C}(\mathbb{R}^d)$ probando con esto el resultado buscado. □

Estamos ya en condiciones de probar el resultado clave del trabajo.

⊙ **Idea intuitiva lema 4.7:**

Afirma que **podemos aproximar cualquier función continua definida con una red neuronal**. Además, si los valores de entrada está dentro de un rango acotado tendremos una cota de error única para todo el dominio.

4. Las redes neuronales son aproximadores universales

Teorema 4.5. Para cualquier función de activación ψ , $d \in \mathbb{N}$ y medida de probabilidad μ en $(\mathbb{R}^d, \mathcal{B}^d)$, se tiene que $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ es uniformemente denso para compactos en $\mathcal{C}(\mathbb{R}^d)$ y denso en $\mathcal{M}(\mathbb{R}^d)$ para a la distancia ρ_μ .

🕒 **Idea intuitiva teorema 4.5:**

Afirma que las **redes neuronales son capaces de aproximar funciones medibles**, es decir funciones muy generales con formas variopintas. La diferencia que guarda con el lema 4.7 es que si lo que estamos aproximando es una función medible, no tendremos una cota global del error, es decir **podemos asegurar que en casi todos los puntos esté muy próximo a la función que queremos aproximar**, pero pudiera ser que al evaluar un punto concreto el error sea muy grande.

🔍 **Reflexión sobre 4.5**

Por lo que acabamos de ver no se aproxima igual de bien las funciones continuas a las medibles. La cuestión es la siguiente ¿la función que nosotros queremos aproximar con una red neuronal es continua o solo medible? A nivel práctico es ante todo desconocida, luego debemos suponer las hipótesis menos restrictivas, es decir, que sea solo medible.

Demostración. En el lema anterior acabamos de ver que $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ es uniformemente denso en compactos en $\mathcal{C}(\mathbb{R}^d)$ y gracias al lema 4.2 (Si $\{f_n\}$ es una sucesión de funciones en $\mathcal{M}(\mathbb{R}^d)$ que converge uniformemente en un compacto a f entonces $\rho_\mu(f_n, f) \rightarrow 0$.) esto implica que $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ sea ρ_μ -denso en $\mathcal{C}(\mathbb{R}^d)$.

Finalmente como consecuencia del lema 4.3 que indica que para cualquier medida finita μ se tiene que $\mathcal{C}(\mathbb{R}^d)$ es denso en $\mathcal{M}(\mathbb{R}^d)$ para la distancia ρ_μ se tiene por la transitividad de ser denso (ver mismo razonamiento que el usado para la demostración del teorema 4.3) que $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ es ρ_μ -denso en $\mathcal{M}(\mathbb{R}^d)$. \square

4.4.1.1. Relevancia del teorema de aproximación de funciones medibles 4.5

Notemos que $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ representa el conjunto de las redes neuronales *feedforward* con tan solo una capa oculta, luego acabamos de probar que las redes neuronales *feedforward* con tan solo una capa oculta son capaces de aproximar cualquier función medible con un error prefijado cualquier función continua y aproximar a secas cualquier función medible con cualquier medida μ , distancia ρ_μ , independiente de la función de activación μ seleccionada (ya sea continua o no) y con cualquier dimensión d del espacio de entrada.

Trataremos ahora de generalizar la tesis expuesta en el teorema 4.5 sobre las funciones medibles. Para ello recordaremos el teorema de Lusin.

Teorema 4.6 (Teorema de Lusin). Si μ es una medida regular de Borel, E un conjunto de medida finita y f una función medible en E entonces para cualquier $\epsilon > 0$ existirá un conjunto compacto K en E tal que $\mu(E \setminus K) \leq \epsilon$ y donde f es continua en K .

Demostración. Demostración en páginas 242 y 243 de [Hal74]. \square

Notemos los puntos clave de este teorema, nos va a permitir *trabajar* con una función medible como si fuera continua en un compacto todo lo parecido a \mathbb{R}^r como se quiera.

Teorema 4.7. (Caracterización de normalidad de Tietze) Sea X un espacio Hausdorff. Son equivalentes las siguientes afirmaciones:

1. X es normal.
2. Para cada conjunto cerrado $A \subset X$ y para cualquier función continua $f : A \rightarrow \mathbb{R}$, f admite una extensión continua $F : X \rightarrow \mathbb{R}$. Además, si para todo $a \in A$ se cumple que $|f(a)| < c \in \mathbb{R}$, se puede elegir F de tal forma que satisfaga que $|F(x)| < c$ para todo $x \in X$.

(Demostración en páginas 149-151 de [Dug66])

Como el ambiente actual en el que estamos trabajando es el espacio $(\mathbb{R}^d, \mathcal{T})$ que sabemos que es normal y puesto que es habitual que nuestras funciones estén definidas en compactos de \mathbb{R}^d , las podremos extender a \mathbb{R}^d .

Corolario 4.3. Para cada función $g \in \mathcal{M}(\mathbb{R}^d)$ existe un subconjunto compacto K de \mathbb{R}^d y $f \in \mathcal{H}(\mathbb{R}^d, \mathbb{R})$ tal que para cualquier $\epsilon > 0$ se tiene que $\mu(K) > 1 - \epsilon$ y para cada $x \in K$ tenemos que

$$|f(x) - g(x)| < \epsilon,$$

independientemente de la función de activación ψ , dimensión d o medida μ .

② **Idea intuitiva corolario 4.3**

Este teorema corrige la carencia sobre la precisión del error que describíamos en la idea intuitiva del teorema 4.5. Podemos encontrar una red neuronal que aproxime cualquier función medible que queramos en todos los puntos del espacio que queramos.

Demostración. Sea $\epsilon > 0$ fijo pero arbitrario. Gracias al teorema de Lusin 4.6 existe un subconjunto compacto $K \subset \mathbb{R}^d$ de medida $\mu(K) > 1 - \epsilon$ donde la restricción $g|_K$ es continua.

Por otra parte, en virtud de la caracterización de Tietze 4.7, por estar $g|_K$ definida en un compacto, admite una extensión continua $G : \mathbb{R}^d \rightarrow \mathbb{R}$ tal que

$$G|_K = g|_K.$$

Por ser G continua en un compacto, por la densidad de las redes neuronales en compactos en $\mathcal{C}(\mathbb{R}^d)$ (lema 4.7) se tiene que existirá $f \in \Sigma\Pi^d(\psi)$ tal que

$$\sup_{x \in K} |G(x) - f(x)| < \epsilon.$$

Por lo que podemos afirmar que para todo $x \in K$

$$|f(x) - g(x)| \leq |f(x) - G(x)| + |G(x) - g(x)| < \epsilon + 0 = \epsilon$$

como queríamos probar. □

4.5. Generalización a espacios L_p

Hasta ahora habíamos considerado el espacio de funciones continuas $\mathcal{C}(\mathbb{R}^d)$ como subespacio dentro del espacio de funciones medibles $\mathcal{M}(\mathbb{R}^d)$. Sin embargo, ser continua es una hipótesis muy estricta ya que existe una amplia gama de subespacios que contienen al de las funciones continuas y están contenidos en el de funciones medibles. Es por ello que vamos a realizar una generalización de los teoremas para espacios L_p . De manera intuitiva estos espacios nos van a permitir considerar funciones que no necesariamente sean continuas y que incluso no están acotadas.

Definición 4.10 (Espacios L_p). Se llama espacio $L_p(\mathbb{R}^d, \mu)$ o simplemente L_p al conjunto de funciones $f \in \mathcal{M}(\mathbb{R}^d)$ tales que

$$\int |f(x)|^p d\mu < \infty.$$

Se define la norma de L_p como

$$\|f\|_p = \left(\int |f(x)|^p d\mu \right)^{\frac{1}{p}}$$

y la distancia asociada al espacio L_p se define como

$$\rho_p(f, g) = \|f - g\|_p.$$

Corolario 4.4. Si existe un subconjunto compacto K en \mathbb{R}^d de medida $\mu(K) = 1$ entonces $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ es ρ_p -denso en $L_p(\mathbb{R}^d, \mu)$ para cualquier $p \in [1, \infty)$, independientemente de ψ, d o μ .

② **Idea intuitiva de espacio L_p**

Con esta definición lo que se trata es de considerar funciones que tomen un valor real en la mayoría de sus puntos. Ya que la integral no varía su valor si se aplican cambios puntuales.

② **Idea intuitiva corolario 4.4**

Se prueba que las redes neuronales son capaces de aproximar cualquier función de la clase recién introducida.

4. Las redes neuronales son aproximadores universales

Demostración. Se quiere probar que para cualquier $g \in L_p$ y $\epsilon > 0$ existe $f \in \mathcal{H}(\mathbb{R}^d, \mathbb{R})$ tal que

$$\rho_p(f, g) < \epsilon.$$

Por pertenecer g a L_p existe una constante M real positiva lo suficientemente grande tal que si definimos la función $h = g1_{|g| < M}$ esta satisface que

$$\rho_p(g, h) < \frac{\epsilon}{3}. \quad (4.10)$$

Además como h es una función acotada de L_p , podemos encontrar una función s continua que es límite de una sucesión de funciones simples (pag 241-242, teoremas 55C y 55D [Hal74]) y la cual cumple que

$$\rho_p(h, s) < \frac{\epsilon}{3}. \quad (4.11)$$

Por el teorema 4.5, al estar en un compacto K y por ser $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ uniformemente denso en compactos hay una $f \in \mathcal{H}(\mathbb{R}^d, \mathbb{R})$ la cual cumple que

$$\sup_{x \in K} |f(x) - s(x)|^p < \left(\frac{\epsilon}{3}\right)^p.$$

Y por hipótesis $\mu(K) = 1$ y definición de la distancia ρ_p se tiene la siguiente desigualdad:

$$\rho_p(f, s) = \left(\int |f(x) - s(x)|^p d\mu \right)^{\frac{1}{p}} \leq \left(\int \left(\frac{\epsilon}{3}\right)^p d\mu \right)^{\frac{1}{p}} = \left(\mu(K) \left(\frac{\epsilon}{3}\right)^p\right)^{\frac{1}{p}} = \frac{\epsilon}{3}. \quad (4.12)$$

Gracias a la desigualdad triangular y las desigualdades (4.12), (4.10) y (4.11) tenemos

$$\rho_p(f, g) \leq \rho_p(f, s) + \rho_p(s, h) + \rho_p(h, g) < \frac{\epsilon}{3} + \frac{\epsilon}{3} + \frac{\epsilon}{3} = \epsilon.$$

Probando con ello lo buscado. □

Corolario 4.5. Si μ es una medida de probabilidad en $[0, 1]^d$ entonces $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ es ρ_p -denso en $L_p([0, 1]^d, \mu)$ para todo $p \in [1, \infty)$, independientemente de ψ, d, μ .

🕒 **Idea intuitiva**
corolario 4.5

A diferencia que con el corolario 4.4, aquí se afirma que las redes neuronales son capaces de aproximar cualquier función de la clase recién introducida cuyo dominio parta de $[0, 1]^d$, es decir se está concretando los valores de entrada que pueden tomar las funciones.

Demostración. Es consecuencia directa del corolario previo 4.4 donde para este caso particular $K = [0, 1]^d$ un compacto de \mathbb{R}^d que cumple que $\mu(K) = 1$. □

Corolario 4.6. Sea μ una medida, que para un conjunto finito de puntos O cumple que $\mu(O) = 1$, entonces, para cualquier función medible $g \in \mathcal{M}(\mathbb{R}^d)$ y sea cual sea $\epsilon > 0$ existe $f \in \mathcal{H}(\mathbb{R}^d, \mathbb{R})$ la cual cumple que

$$\mu\{x : |f(x) - g(x)| < \epsilon\} = 1.$$

Demostración. Por el teorema 4.5 existe $f \in \mathcal{H}(\mathbb{R}^d, \mathbb{R})$ tal que para cualquier $\epsilon_1, \epsilon_2 > 0$ se cumple que $\mu\{x : |f(x) - g(x)| > \epsilon_1\} < \epsilon_2$. Sea O el conjunto de puntos tal que $\mu(O) = 1$. Por ser finito O podemos encontrar

$$\delta = \min_{x \in O} \{\mu(x) : \mu(x) > 0\}. \quad (4.13)$$

Sin pérdida de generalidad tomamos $\epsilon < \delta$ y entonces para que la f fijada cumpla que

$$\rho_\mu(f, g) = \epsilon$$

debe de cumplirse que

$$\rho_\mu(f, g) = \inf\{\epsilon_1 > 0 : \mu\{x : |f(x) - g(x)| > \epsilon_1\} < \epsilon_1\} = \epsilon,$$

pero por cómo tomamos ϵ en (4.13) se tiene que $\mu\{x : |f(x) - g(x)| > \epsilon\} = 0$.

Por lo que acabamos de probar, como queríamos, que

$$\mu\{x : |f(x) - g(x)| < \epsilon\} = 1.$$

□

Nótese que con este corolario lo que se está indicando es que dado un conjunto finito y sus respectivas imágenes, se puede encontrar una red neuronal que para tales puntos *devuelva* el valor exacto.

Definición 4.11 (Función Booleana). Decimos que una función es Booleana si su dominio es $\{0, 1\}^d$ para algún $d \in \mathbb{N} \setminus \{0\}$ y su codominio es $\{0, 1\}$. Es decir,

$$f : \{0, 1\}^d \longrightarrow \{0, 1\}.$$

Ejemplos conocidos son la función *or* : $\{0, 1\}^d \longrightarrow \{0, 1\}$ que vale uno si alguno de su entrada es uno y la función *and* : $\{0, 1\}^d \longrightarrow \{0, 1\}$ que se define como $and(x_1, \dots, x_d) = \prod_{i=1}^d x_i$.

Corolario 4.7. Para cada función Booleana $g : \{0, 1\}^d \longrightarrow \{0, 1\}$ y cada $\epsilon > 0$ existe una red neuronal $f \in \mathcal{H}(\mathbb{R}^d, \mathbb{R})$ tal que

$$\max_{x \in \{0, 1\}^d} |g(x) - f(x)| < \epsilon.$$

Demostración. Se define la función $\mu : \mathbb{R}^d \longrightarrow [0, 1]$ de forma que

$$\mu(x) = \begin{cases} \frac{1}{2^d} & \text{si } x \in \{0, 1\}^d \\ 0 & \text{si } x \notin \{0, 1\}^d \end{cases}$$

Se tiene que μ es una medida ya que cumple que

1. Hipótesis de acotación: $0 \leq \mu(A) \leq 1$ para $A \in \mathcal{P}(\mathbb{R}^d)$.
2. La probabilidad del vacío es nula y la del total es la unidad.
3. La probabilidad de la unión es la suma de la probabilidades.

$$P(\cup_{i=1}^n A_i) = \sum_{i=1}^n P(A_i). \quad \forall A_i \in \mathcal{P}(\mathbb{R}^d).$$

Ⓢ **Relevancia práctica del corolario 4.7**

Este resultado nos permite **aproximar funciones que actúen como clasificadores discretos**. Por ejemplo sea g como una función que dada una imagen x indica si hay un perro haciendo valer $g(x) = 1$ y en caso contrario $g(x) = 0$.

Ⓢ **Observación sobre la implementación del corolario 4.7**

El resultado nos indica que podemos obtener una red neuronal h que aproxime tal clasificador, pero **tal red neuronal no necesariamente tomará valores discretos**, es decir, pudiera darse el caso que $h(\{x : g(x) = 0\}) \subset [-0.2, 0.3]$ y que $h(\{x : g(x) = 1\}) \subset [0.9, 1.2]$, por lo que se pone de manifiesto en este resultado, que en caso de requerirse de una salida completamente discreta debería de componerse con otra función θ tal que $\theta \circ h(\{x : g(x) = 0\}) = 0$ y $\theta \circ h(\{x : g(x) = 1\}) = 1$.

4. Las redes neuronales son aproximadores universales

Como la cardinalidad de $\{0, 1\}^d$ es 2^d podemos aplicar el corolario 4.6 y entonces sabemos que existe $f \in \mathcal{H}(\mathbb{R}^d, \mathbb{R})$ tal que

$$\mu\{x : |f(x) - g(x)| < \epsilon\} = 1,$$

es decir que

$$\max_{x \in \{0,1\}^d} |g(x) - f(x)| < \epsilon$$

como queríamos probar. □

⊗ La necesidad de que θ podría indicar una deficiencia en la estructura de construcción ¿y si aproximáramos directamente con enteros? Ejemplo de algo parecido [?]

Lema 4.8. Si una función de activación ψ alcanza el cero y el uno, esto es si existen dos constantes reales M_1, M_2 tales que

$$\psi(M_1) = 0 \text{ y } \psi(M_2) = 1$$

entonces existe una constante real positiva M tal que

$$\psi(-M) = 1 - \psi(M) = 0.$$

Demostración. Sea $M = \max\{|M_1|, |M_2|\}$ y por ser ψ una función de activación sabemos que es no decreciente y que su imagen pertenece al intervalo $[0, 1]$.

Por tanto

$$\begin{aligned} 0 &\leq \psi(-M) \leq \psi(M_1) = 0 & \text{luego } \psi(-M) &= 0, \\ 1 &\geq \psi(M) \geq \psi(M_2) = 1 & \text{luego } \psi(M) &= 1 \end{aligned}$$

Gracias a estas desigualdades es fácil ver que

$$\psi(-M) = 1 - \psi(M) = 0$$

como queríamos probar. □

Es interesante percatarse de que de no exigirse la hipótesis de que ψ alcanza el cero y el uno no puede asegurarse la igualdad demostrada. Pongamos como ejemplo la siguiente función de activación

$$\psi(x) = \begin{cases} 0 & \text{si } x \leq 0 \\ \frac{|x|}{1+|x|} & \text{si } 0 < x \end{cases}$$

Lema 4.9. Dado un conjunto finito de vectores $\Lambda \subset \mathbb{R}^d$ con d natural positivo. Existe un vector $p \in \mathbb{R}^d$ que satisface que para cualesquiera $x, y \in \Lambda$ diferentes

$$p \cdot (x - y) \neq 0.$$

⊗ **Idea clave teorema 4.8**

Podemos conseguir una red neuronal que valga lo que queramos en un conjunto finito de puntos.

Demostración. Si n es el cardinal de Λ , consideramos el conjunto U definido como la unión de $n(n-1)$ hiperplanos de \mathbb{R}^d

$$U = \bigcup_{\substack{x, y \in \Lambda \\ x \neq y}} \{p \in \mathbb{R}^d : p \cdot (x - y) = 0\}.$$

Puesto que \mathbb{R}^d no puede ser expresado como unión finita de hiperplanos, $U \subsetneq \mathbb{R}^d$ y por tanto existirá $p \in \mathbb{R}^d \setminus U$ tal que

$$p \cdot (x - y) \neq 0 \text{ para cualesquiera } x, y \in \Lambda \text{ diferentes,}$$

como queríamos probar. □

Teorema 4.8 (Sobre el entrenamiento práctico de redes neuronales). *Sea $\Lambda = \{x_1, \dots, x_n\}$ un conjunto de puntos distintos de \mathbb{R}^d y sea $g : \mathbb{R}^d \rightarrow \mathbb{R}$ una función arbitraria. Si ψ alcanza el cero y el uno, entonces existe una red neuronal $f \in \mathcal{H}(\mathbb{R}^d, \mathbb{R})$ con n neuronas ocultas tal que*

$$f(x_i) = g(x_i) \text{ para todo } i \in \{1, \dots, n\}.$$

Demostración. Con el fin de facilitar la comprensión dividiremos la demostración en dos casos, primero uno particular, cuando $d = 1$ y después el caso general.

Caso primero

Suponemos que $\{x_1, \dots, x_n\} \subset \mathbb{R}$ y tras renombrar podemos suponer que

$$x_1 < x_2 < \dots < x_n.$$

Por alcanzar la función de activación ψ el cero y el uno, gracias al lema 4.8 existe una constante M tal que $\psi(-M) = 1 - \psi(M) = 0$.

Definiremos de manera recursiva la red neuronal buscada f_n .

- Red neuronal f_1 .

Sea A_1 la función afín constante $A_1 = M$. Fijamos $\beta_1 = g(x_1)$. De esta manera la red neuronal f_1 queda definida como $f_1(x) = \beta_1 \psi(A_1(x))$.

- Red neuronal f_k con $1 < k \leq n$.

Se define A_k como la única función afín que cumple que

$$A_k(x_{k-1}) = -M \quad \text{y} \quad A_k(x_k) = M.$$

Fijamos $\beta_k = g(x_k) - g(x_{k-1})$. La red neuronal f_k se calcula como

$$f_k(x) = \sum_{j=1}^k \beta_j \psi(A_j(x)) = (g(x_k) - g(x_{k-1})) \psi(A_k(x)) + f_{k-1}(x).$$

Observemos que así construida se tiene que para cualquier $y > x_k$ la evaluación con la red neuronal resulta $f_k(y) = g(x_k)$.

Veamos por inducción sobre n que así definida para cualquier $1 \leq i \leq n$ se tiene que $f_n(x_i) = g(x_i)$. Además de que tiene un total de n neuronas ocultas.

- Caso base, $n = 1$.

$$f_1(x_1) = \beta_1 \psi(A_1(x_1)) = g(x_1) \psi(M) = g(x_1).$$

Ⓢ **Idea de la demostración del teorema 4.8**

Es interesante reparar en que la demostración se basa en añadir una neurona por cada punto que queramos que tome un valor concreto, esa neurona se activará (es decir, no será nula) cuando la entrada x sea mayor que el valor que la activa x_i y vale la diferencia con el valor anterior x_{i-1} , es decir $g(x_i) - g(x_{i-1})$, como el nodo x_{i-1} también se activará por ser menor menor, el término $g(x_{i-1})$ se suma a la salida de la red y así como una serie telescópica al final solo resultará el valor $g(x_i)$.

Q **Nueva hipótesis de optimización**

El teorema 4.8 nos brinda una heurística de inicialización de los pesos de una red neuronal. La cual se tratará en la sección 5.4.

4. Las redes neuronales son aproximadores universales

- Supuesto que es cierto para $n - 1$ veamos que lo es para n . Evaluación de x_n

$$\begin{aligned} f_n(x_n) &= (g(x_n) - g(x_{n-1}))\psi(A_n(x_n)) + f_{n-1}(x_n) \\ &= (g(x_n) - g(x_{n-1}))\psi(M) + g(x_{n-1}) \\ &= (g(x_n) - g(x_{n-1})) + g(x_{n-1}) \\ &= g(x_n). \end{aligned}$$

Evaluación de x_i con $1 \leq i < n$.

Usando que $0 \leq \psi(A_n(x_i)) < \psi(A_n(-M)) = 0$ y la hipótesis de inducción se tiene que

$$\begin{aligned} f_n(x_i) &= (g(x_n) - g(x_{n-1}))\psi(A_n(x_i)) + f_{n-1}(x_i) \\ &= 0 + g(x_i) \\ &= g(x_i). \end{aligned}$$

Acabamos de probar por inducción que $f_n(x) = g(x)$ para cualquier $x \in \Lambda$, lo que termina la demostración.

Caso segundo

Se tiene para este caso que $\Lambda \subset \mathbb{R}^d$ con $d > 1$. Seleccionamos $p \in \mathbb{R}^d$ cumpliendo que para cualesquiera $x, y \in \Lambda$ distintos $p(x - y) \neq 0$ (es posible de encontrar por el lema 4.9). Gracias a esta condición cada producto $p \cdot x$ con $x \in \Lambda$ es distinto y podemos establecer con ello una relación de orden, que tras renombrar los elementos de Λ queda

$$p \cdot x_1 < p \cdot x_2 < \dots < p \cdot x_n,$$

y como procedimos en el caso primero, definimos de manera recursiva la red neuronal f_n buscada:

- Red neuronal f_1 .

Sea B_1 la función afín de \mathbb{R} a \mathbb{R} constante $B_1 = M$. Fijamos $\beta_1 = g(x_1)$. De esta manera la red neuronal f_1 definida como $f_1(x) = \beta_1\psi(B_1(p \cdot x))$.

- Red neuronal f_k con $1 < k \leq n$.

Se define B_k como la única función afín de \mathbb{R} en \mathbb{R} que cumple que

$$B_k(p \cdot x_{k-1}) = -M \quad \text{y} \quad B_k(p \cdot x_k) = M.$$

Podemos definir entonces $A \in \mathcal{A}(\mathbb{R}^d)$ por $A_k(x) = B_k(p \cdot x)$. Fijamos $\beta_k = g(x_k) - g(x_{k-1})$. La red neuronal f_k se calcula como

$$\begin{aligned} f_k(x) &= \sum_{j=1}^k \beta_j \psi(B_j(p \cdot x)) = (g(x_k) - g(x_{k-1}))\psi(B_k(p \cdot x)) + f_{k-1}(x) \\ &= \sum_{j=1}^k \beta_j \psi(A_j(x)) = (g(x_k) - g(x_{k-1}))\psi(A_k(x)) + f_{k-1}(x) \end{aligned}$$

Así definida, la prueba por inducción es idéntica a la del caso primero y hemos encontrado por tanto la red neuronal f_n buscada. \square

4.5.1. Reflexión sobre el número de neuronas

Ante la pregunta natural de *¿cuántas neuronas son necesarias?* el recién probado teorema nos responde que si estamos entrenando con n datos, con n neuronas es suficiente para volver a reproducir esos datos. Pero esto carece de sentido a nivel práctico por los siguientes motivos.

Naturaleza de los datos

Recordemos que el problema al que nos enfrentamos es el siguiente: queremos ser capaces de predecir cierto fenómeno regido por g una *función* desconocida. Para ello tenemos un *conjunto de muestras* compuestas por pares $(x, g(x))$, es decir, ante la situación x hemos observado que el fenómeno se comporta como $g(x)$. Todo esta situación experimental puede producir incoherencias teóricas, como por ejemplo que se tengan dos muestras $(x_1, y_1), (x_2, y_2)$ tales que $x_1 = x_2$ pero $y_1 \neq y_2$ o que la medición contenga errores, es decir $(x, g(x) + \delta)$.

Se podría paliar esta situación con un preprocesado de los datos.

Naturaleza de la regla subyacente

Supongamos que los datos de entrenamiento son perfectos. Existen infinitas aplicaciones que evalúan de la misma manera un conjunto finito de puntos. ¿Cuáles tomar?

Podríamos, siguiendo el principio de economía de Ockham optar por modelos que reduzcan el número de neuronas, pero entonces la pregunta sería *¿qué número mínimo de neuronas serían necesarios para representar el modelo?* Una solución sería tomar un número de neuronas menor que el tamaño del conjunto de entrenamiento y utilizar esa *redundancia* de datos para *afinar*.

Coste computacional inasumible

Supongamos que los datos son idílicos, el modelo se conoce y se establece un tamaño de datos de entrenamiento suficiente y un número de neuronas acorde ¿podríamos asumir el coste computacional?

4.6. Generalización para multi-output neuronal networks

En las secciones anteriores se han provisto resultados para redes neuronales de salida real. Vamos a generalizar los resultados vistos para ser capaces de aproximar funciones continuas o medibles de \mathbb{R}^d a \mathbb{R}^s con $d, s \in \mathbb{N}$.

Denotaremos por $\mathcal{C}(\mathbb{R}^d, \mathbb{R}^s)$ al conjunto de funciones continuas definidas de \mathbb{R}^d a \mathbb{R}^s y al de funciones medibles de \mathbb{R}^d a \mathbb{R}^s por $\mathcal{M}(\mathbb{R}^d, \mathbb{R}^s)$. La distancia asociada a estos espacios se define como

$$\rho_\mu^s(f, g) = \sum_{i=1}^s \rho_\mu(f_i, g_i).$$

Con la siguiente definición buscamos abstraer el modelo de una red neuronal de una capa oculta y salida múltiple.

Nótese que los vectores $(w_{0i}, w_{1i}, \dots, w_{ri})$ representan a la aplicación afín $A_i((x_1, x_2, \dots, x_r)) = w_{0i} + \sum_{j=1}^d w_{ji}x_j$ con $i \in \{1, \dots, h\}$.

4. Las redes neuronales son aproximadores universales

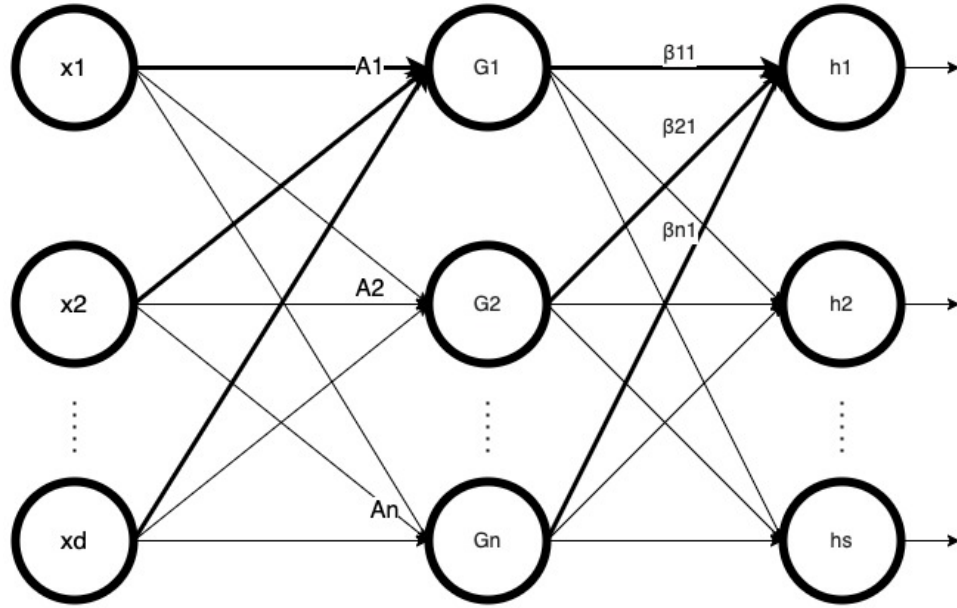


Figura 4.7.: Ejemplo de red neuronal de una capa oculta con n nodos, de dimensión de entrada d y salida s .

Definición 4.12 (Abstracción de una red neuronal con una capa oculta y múltiple salida). Para cualquier función Borel medible G , definida de \mathbb{R} a \mathbb{R} y cualquiera naturales positivo $d, s \in \mathbb{N}$ se define a la clase de funciones $\mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$ como

$$\mathcal{H}(\mathbb{R}^d, \mathbb{R}^s) = \{h : \mathbb{R}^d \rightarrow \mathbb{R}^s, h = (h_1, h_2, \dots, h_s) /$$

$$\text{con } h_i : \mathbb{R}^d \rightarrow \mathbb{R}, h_i(x) = \sum_{j=1}^{n_i} (\beta_{ji} G(A_j(x))) \quad i \in \{1, 2, \dots, s\},$$

$$x \in \mathbb{R}^d, \beta_{ji} \in \mathbb{R}, A_j \in \mathcal{A}(\mathbb{R}^d), n_i \in \mathbb{N}, i \in \{1, \dots, s\}\}.$$

🕒 **Por qué se ha destacado la observación** Se destaca tal observación para remarcar que una red neural con salidas en varias dimensiones tiene una definición mucho más estricta y que podría dar a confusión.

🕒 **Observación:**

Es interesante apreciar que se tiene que

$$\mathcal{H}(\mathbb{R}^d, \mathbb{R}^s) \subsetneq \{h : \mathbb{R}^d \rightarrow \mathbb{R}^s, h = (h_1, h_2, \dots, h_s) /$$

$$h_i \in \mathcal{H}(\mathbb{R}^d, \mathbb{R}) \quad \forall i \in \{1, \dots, s\}\}. \quad (4.14)$$

aunque como veremos en el corolario 4.8 $\mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$ es un espacio denso en el conjunto que acabamos de presentar. Si tenemos presente que de acorde a nuestras definiciones una red neuronal h_i viene determinada por un conjunto de funciones afines $A_j^{(i)}$ y escalares $\beta^{(i)j}$, se daría la igualdad en 4.14 si imponemos que para cualquier par de redes neuronales h_k, h_j se satisface que $A_i^{(k)} = A_i^{(j)}$.

No es difícil pensar que su versión generalizada sea:

Definición 4.13. Dadas las mismas hipótesis que en la definición anterior, se define la si-

guiente clase de funciones como

$$\sum \prod^{d,s}(G) = \{f : \mathbb{R}^d \longrightarrow \mathbb{R}^s, f = (f_1, f_2, \dots, f_s) /$$

$$\text{con } f_i : \mathbb{R}^d \longrightarrow \mathbb{R}, f_i(x) = \sum_{j=1}^h \left(\beta_{ji} \prod_{k=1}^{l_{ji}} G(A_{jk}(x)) \right) \quad i \in \{1, 2, \dots, s\},$$

$$x \in \mathbb{R}^d, \beta_{ji} \in \mathbb{R}, A_{jk} \in \mathcal{A}(\mathbb{R}^d); h, l_{ji} \in \mathbb{N}\}.$$

Corolario 4.8. Los teoremas 4.4, 4.5 y los corolarios 4.3, 4.4, 4.5, 4.6 y 4.7 permanecen válidos si se sustituye $\mathcal{H}(\mathbb{R}^d, \mathbb{R})$ por $\mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$, $\sum \prod^d(\psi)$ por $\sum \prod^{d,s}(\psi)$, los espacios de funciones continuas y medibles por $\mathcal{C}(\mathbb{R}^d, \mathbb{R}^s)$ y $\mathcal{M}(\mathbb{R}^d, \mathbb{R}^s)$ respectivamente.

Demostración. Observemos que todos los teoremas y lemas mencionados basan su tesis en la existencia de una red neuronal es decir, que si llamamos según convenga $\mathcal{F}^{d,s}$ a $\mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$ o $\sum \prod^{d,s}(\psi)$ deberemos de encontrar $f \in \mathcal{F}^{d,s}$ que cumplan las respectivas tesis para salidas múltiples.

La prueba se construirá por inducción sobre el número de salidas s .

El caso base $s = 1$ viene dado por los respectivos teoremas y lemas ya probados. Supuesto cierto para $s = n$ veamos que se cumple para $s = n + 1$:

Se quiere encontrar $f = (f_1, f_2, \dots, f_n, f_{n+1})$ de $n + 1$ salidas, por hipótesis de inducción existe $g_n \in \mathcal{F}^{d,n}$ con $g_n = (f_1, f_2, \dots, f_n)$ y con h_n sumandos. Denotamos a los pesos de las transformaciones afines $w_{ij} \in \mathbb{R}$ con $i \in \{0, 1, \dots, d\}$ y $j \in \{1, \dots, h_n\}$ y $\beta_{kl} \in \mathbb{R}$ con $k \in \{1, \dots, h_n\}$ y $l \in \{1, \dots, n\}$.

También existe $g_1 \in \mathcal{F}^{d,1}$ cumpliendo que $g_1 = f_{n+1}$ con h_1 sumandos en la capa oculta y pesos $w'_{ij} \in \mathbb{R}$ con $i \in \{0, 1, \dots, d\}$ y $j \in \{1, \dots, h_1\}$ y $\beta'_{kl} \in \mathbb{R}$ con $k \in \{1, \dots, h_1\}$ y $l = n + 1$ (Ver figura 4.7 para orientarse en la notación tomada).

Considerando f compuesta por $h_n + h_1$ sumandos y donde sus pesos son los siguientes:

El peso \tilde{w} de las funciones afines: Para cuales quiera $i \in \{0, 1, \dots, d\}$ y $j \in \{1, \dots, h_n, h_n + 1, \dots, h_n + h_1\}$ determinaremos la siguiente casuística

1. Si $1 \leq j \leq h_n$ entonces $\tilde{w}_{ij} = w_{ij}$.
2. Si $h_n < j \leq h_n + h_1$ entonces $\tilde{w}_{ij} = w_{i(j-h_n)}$.

Para los pesos $\tilde{\beta}$, para cualquier $k \in \{1, \dots, h_n, h_n + 1, \dots, h_n + h_1\}$ y $l \in \{1, \dots, n + 1\}$:

1. Si $k \in \{1, \dots, h_n\}$ y $l \in \{1, \dots, n\}$ entonces $\tilde{\beta}_{kl} = \beta_{kl}$.
2. Si $k \in \{1, \dots, h_n\}$ y $l = n + 1$ entonces $\tilde{\beta}_{kl} = 0$.
3. Si $k \in \{h_n + 1, \dots, h_n + h_1\}$ y $l \in \{1, \dots, n\}$ entonces $\tilde{\beta}_{kl} = 0$.
4. Si $k \in \{h_n + 1, \dots, h_n + h_1\}$ y $l = n + 1$ entonces $\tilde{\beta}_{kl} = \beta'_{(k-h_n)l}$.

Notemos que $f = (f_1, f_2, \dots, f_n, f_{n+1})$, es decir $f \in \mathcal{F}^{d,s}$, y que para cada teorema o lema cada una de las proyecciones de f cumple la tesis, es decir f cumple lo buscado. \square

🔑 **Idea clave corolario 4.8**

En esencia, todos los resultados probados hasta ahora para redes neuronales con salida real de dimensión uno son válidos para cualquier tamaño de salida, es decir, **lo que a nosotros nos interesa podemos aproximar cualquier función medible que vaya entre espacios de dimensión finita.**

🔑 **Idea clave demostración corolario 4.8**

La demostración del corolario es totalmente constructiva: si se desea una red neuronal de salida $s \in \mathbb{N}$, se construyen de manera independiente s redes neuronales de salida de dimensión uno (una para cada salida buscada) y las concatenamos haciendo valer cero las conexiones entre nodos que no pertenecieran a las redes de partida.

🔍 **Nueva hipótesis de optimización**

La demostración del corolario nos da una **técnica constructiva** de obtener redes neuronales de varias salidas, que puede ser de valía para aplicarla como **heurística para inicializar los pesos de la red neuronal** como ya apuntábamos en las notas del teorema 4.8.

Conclusión sobre el teorema anterior

A la vista de la demostración constructiva se nos acaba de decir de manera indirecta que si queremos construir una red neuronal a partir de un tamaño de conjunto entrenamiento E y de salida de dimensión s , con una red neuronal de $E \times s$ neuronas en la ocultas será suficiente y para este caso la reflexión expuesta en la sección 4.5.1 es idéntica.

4.7. Consideración sobre la capacidad de cálculo

Suele pasar peligrosamente desapercibido que el teorema 4.8 recién probado asegura que se podrá encontrar una red neuronal en $\mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$ que aproxime todo lo bien que queramos la función ideal; sin embargo, destaquemos que los parámetros que caracterizan a la red neuronal encontrada son reales. Es decir, si nuestra red neuronal toma valores irracionales difícilmente será computable en un ordenador por su infinitud.

Esto nos impone una nueva restricción en el espacio de búsqueda, ya que no solo debe de reducirse el error, si no que los parámetros que representan la red deben de estar limitados a cierta precisión: la propia de un ordenador.

Comenzaremos viendo que en efecto una red neuronal con parámetros en el cuerpo de los enteros es factible como aproximador universal.

Aportación original

Teorema 4.9. *El espacio $\mathcal{H}(\mathbb{Q}^d, \mathbb{Q}^s)$ es denso al espacio $\mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$.*

Demostración. Se quiere probar que para todo $h^r \in \mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$, dado cualquier $\epsilon \in \mathbb{R}^+$ existirá $h^q \in \mathcal{H}(\mathbb{Q}^d, \mathbb{Q}^s)$ tal que $\rho_\mu(h^r, h^q) < \epsilon$.

La red neuronal h^r está determinada por un conjunto finito de parámetros supongamos que hay q y que están determinados por un índice del conjunto Λ .

Sea $\alpha_i^0 \in \mathbb{R}$, el primer índice h^r definimos la red neuronal h^1 con coeficientes α_i^1 con $i \in \Lambda$ de tal forma que los parámetros que la determinan vienen dados

$$\alpha_i^1 = \alpha_i^0 \text{ con } i \in \Lambda \text{ y } i \neq 1.$$

El primer parámetro α_1^1 , será un número racional seleccionado de tal forma que se satisfaga que

$$\rho_\mu(h^r, h^1) < \frac{\epsilon}{q}.$$

Por la densidad de los racionales en los reales sabemos que α_1^1 .

Siguiendo la misma idea, para $j \in \{2, \dots, q\}$ se define la red neuronal h^j cuyos parámetro vienen dados por

$$\alpha_i^j = \alpha_i^{j-1} \text{ con } i \in \Lambda \text{ y } i \neq j;$$

y $\alpha_j^j \in \mathbb{Q}$ seleccionado de tal forma que

$$\rho_\mu(h^j, h^{j-1}) < \frac{\epsilon}{q}.$$

De esta forma, en virtud de la desigualdad triangular, se ha encontrado una red neuronal h^q que todos sus parámetros son racionales y satisface $\rho_\mu(h^r, h^q) < \epsilon$. \square

Si bien los números racionales tienen el potencial de ser computados por su finitud, seguimos sujetos a que el número de decimales y tamaño de su parte entera *sean lo suficientemente pequeños* como para poder ser representados y calculados con un ordenador. Esto no tiene que ser, cierto y de hecho abre una nueva cuestión: Al aumentar el número de neuronas, ¿la precisión demandada por una red neuronal también disminuye?

De ser cierto este resultado, podría empezar a denominarse el espacio de las redes neuronales de n neuronas y d bits de precisión.

Es más, una vez limitados los decimales con los que se puede representar una red neuronal; puesto que los números racionales son isomorfos a los enteros, se podría plantear establecer un isomorfismo entre las redes neuronales con parámetros en los racionales y el y el de las redes neuronales con entradas en los enteros.

Todas estas pesquisas tienen su interés ya que por las arquitecturas actuales, los números flotantes (rationales con un límite de decimales) se calculan en la GPUs, mientras que los enteros en las CPUs, siendo más rápidas la segundas⁴ [LKC⁺10].

⁴En el blog del investigador Long Zhou, se comenta se da una visión favorable sobre las CPUs y cómo en la actualidad se comenten errores a la hora de comparar las GPUs y CPUs, dejo link a la publicación. Consultada por última vez el 23 de mayo del 2022, URL: <https://long-zhou.github.io/2013/02/12/CPU-GPU-comparison.html>

5. Construcción técnica de las redes neuronales de una sola capa

Vista la formulación teórica de una red neuronal de una sola capa introducida en 4.1 explicaremos a continuación una construcción técnica junto con un análisis del costo necesario. Particularmente el algoritmo presentado es una ligera modificación del algoritmo de *forward propagation* explicado en [Biso6], ha sido necesaria su modificación para ser totalmente fieles a nuestro enfoque teórico.

Además, puesto que nuestro objetivo es optimizar seremos muy meticulosos en cuanto a analizar el coste computacional tanto de cómputo como de memoria.

5.1. Componentes de una red neuronal de una capa oculta

Como ya habíamos definido en 4.1 para nosotros una red neuronal será una función $h : X \rightarrow Y$ con $X \subseteq \mathbb{R}^d, Y \subseteq \mathbb{R}^s$ cuya proyección k -ésima viene dada por

$$h_k(x) = \sum_{i=1}^n \beta_{ik} \gamma_i(A_i(x)) = \sum_{i=1}^n \beta_{ik} \gamma_i \left(w_{0i} + \sum_{j=1}^d w_{ji} x_j \right)$$

donde n es el número de neuronas, $\gamma_i \in \Gamma$, funciones medibles definidas de \mathbb{R} a \mathbb{R} , $\beta_{ik} \in \mathbb{R}$ y $A_i \in \mathcal{A}(\mathbb{R}^d)$.

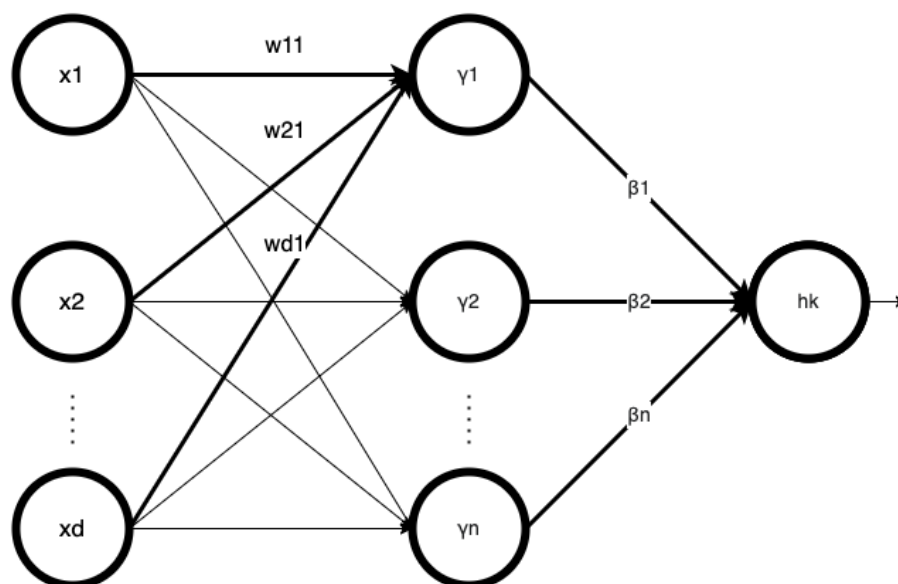


Figura 5.1.: Grafo de una red neuronal de una capa oculta

5. Construcción técnica de las redes neuronales de una sola capa

Ante esta definición el número de parámetros a ajustar es:

- De la forma β_{ik} : ns .
- De la forma w_{ij} : $n(d+1)$.

Lo que hace un total de $n(s+d+1)$ parámetros, donde n es el número de neuronas, s la dimensión de salida y d la dimensión de entrada. Por lo general d y s son fijos ya que se suponen requisitos del problema, luego si se desea reducir el coste en memoria deberá de hacerse disminuyendo el número de neuronas.

Analizaremos más a fondo sus componentes.

Construcción de la primera capa

La primera capa está compuesta por el conjunto de n combinaciones lineales del vector de entrada (x_1, \dots, x_d) a las cuales denominaremos *activaciones*, n será el número de sumandos definidos en 4.1 que equivale al número de neuronas en la capa oculta.

$$a_i = w_{0i} + \sum_{j=1}^d w_{ji}x_j \text{ con } j \in \{1, \dots, n\}.$$

Nos referiremos a los parámetros w_{ji} como *pesos* y al parámetro w_{0i} como *sesgo*.

La memoria necesaria será $(d+1)n\mathfrak{F}$ con \mathfrak{F} es el espacio requerido por un peso. El coste computacional es además de d multiplicaciones y $d+1$ sumas.

Unidades ocultas

Cada una de esas *activaciones* será transformada utilizando una *función de activación* γ_j

$$z_i = \gamma_i(a_i).$$

En el contexto de las redes neuronales a z_i se le conoce como *unidad oculta*. Ésta podría ser de nuevo transformada por una combinación lineal, en nuestro caso tan solo por el producto de un escalar, ya que como se adelantó en la sección 4.1, frente a la transformación afín usualmente propuesta, esta transformación no es necesaria para asegurar la convergencia, profundizaremos sobre esto más adelante.

Finalmente la salida vendrá dada por

$$h_k = \sum_{i=1}^n \beta_{ik}z_i \text{ con } k \in \{1, \dots, s\}.$$

Nótese que ahora el tamaño de variables de entrada es M y hay un total de s unidades de activación, tanto M como s son valores fijados por el diseñador de la red ya que a priori no tenemos otra información.

5.1.1. Consideraciones sobre la irrelevancia del sesgo

Aportación original

Dados $X \subseteq \mathbb{R}^d$, $Y \subseteq \mathbb{R}^s$ y Γ un conjunto no vacío de funciones medibles definidas de \mathbb{R}

a \mathbb{R} , denotaremos como $\mathcal{H}^+(X, Y)$ al conjunto de redes neuronales a las cuales se le ha añadido un sesgo.

$$\mathcal{H}^+(X, Y) = \{h : X \rightarrow Y \mid h_k(x) = \sum_{i=1}^n (\beta_{ik} \gamma_i(A_i(x)) + \alpha_{ik}),$$

donde h_k es la proyección k -ésima de h con $k \in \{1, \dots, s\}$,
 $n \in \mathbb{N}$, $\gamma_i \in \Gamma$, $\beta_{ik} \in \mathbb{R}$ y A_i una aplicación afín de \mathbb{R}^d a \mathbb{R} .

Está claro que al introducir tal sesgo se añade en memoria un coste de $n\mathfrak{F}$ con n el número de neuronas en la capa oculta y que además el costo de cómputo se ve aumentado en la misma proporción. Sin embargo podría obtenerse un beneficio en cuanto a precisión, vamos a proceder a analizar esta idea. Es evidente que

$$\mathcal{H}(X, Y) \subseteq \mathcal{H}^+(X, Y) \quad (5.1)$$

Además al estar trabajando con una sola capa, se tiene que para cualquier $h^+ \in \mathcal{H}^+(X, Y)$

$$h^+ = \sum_{i=1}^n (\beta_{ik} \gamma_i(A_i(x)) + \alpha_{ik}) = \sum_{i=1}^n (\beta_{ik} \gamma_i(A_i(x))) + O,$$

con $O \in \mathbb{R}$ un parámetro libre.

Ante esto, al igual que se hacía en el lema 4.5 es fácil obtener una neurona de valor constante y por tanto, para un conjunto fijo de neuronas n , se tiene que

$$\mathcal{H}_n^+(X, Y) \subsetneq \mathcal{H}_{n+1}(X, Y),$$

de esta relación se obtienen dos cosas: puesto que n era arbitrario y

$$\mathcal{H}(X, Y) = \bigcup_{n \in \mathbb{N}} \mathcal{H}_n(X, Y)$$

entonces como espacios de funciones

$$\mathcal{H}(X, Y) = \mathcal{H}^+(X, Y).$$

Por otra parte, la precisión que pueda aportar el sesgo es superada añadiendo una neurona más a un modelo sin sesgo, es más, también se obtiene una mejora en memoria, ya que $\mathcal{H}_n^+(X, Y)$ requiere de $n\mathfrak{F}$ espacio de memoria adicional con respecto a $\mathcal{H}_n(X, Y)$ mientras que $\mathcal{H}_{n+1}(X, Y)$ de $(d+1)\mathfrak{F}$ y por lo visto en el teorema de convergencia universal 4.1, la precisión se consigue añadiendo neuronas (la dimensión de los datos es fija), luego podemos suponer que n será mayor que $d+1$.

Hasta ahora hemos comparado la capacidad de expresión por la forma de los elementos de los conjuntos, para comparar su bondad aproximando, vamos a fijar un número de neuronas en la capa oculta n y una función de error cualquiera que mida el error dentro del conjunto de entrenamiento $E_{\mathcal{D}} : \mathcal{H}^+(X, Y) \rightarrow \mathbb{R}_0^+$.

5. Construcción técnica de las redes neuronales de una sola capa

Todas las normas en \mathbb{R}^n son equivalentes luego no hay pérdida de generalidad fijando una cualquiera.

Definimos también el error dentro del espacio Λ como

$$\mathcal{E}_{\mathcal{D}}(\Lambda) = \inf\{E_{\mathcal{D}}(h) : h \in \Lambda\}.$$

Está claro por la relación (5.1) que

$$\mathcal{E}_{\mathcal{D}}(\mathcal{H}^+(X, Y)) \leq \mathcal{E}_{\mathcal{D}}(\mathcal{H}(X, Y))$$

La clave ahora reside en si se satisface la desigualdad opuesta Es decir, dada cualquier $h^+ \in \mathcal{H}_n^+(X, Y)$ con un error de $E_{\mathcal{D}}(h^+)$ existe $h \in \mathcal{H}_n(X, Y)$ tal que $E_{\mathcal{D}}(h) \leq E_{\mathcal{D}}(h^+)$.

Esto no es posible y para darse cuenta basta con considerar el caso de una neurona, con $G(x)$ su función de activación y $k \in \mathbb{R}^+$ definimos la función ideal como $f(x) = G(x) + k$. Tomando tan solo dos puntos convenientemente seleccionados (por ejemplo M y $-M$ tal que $G(-M) = 0$ y $G(M) = 1$) aprecia que $H_1(\mathbb{R}, \mathbb{R})$ no puede aproximar f y sin embargo $f \in H_1^+(\mathbb{R}, \mathbb{R})$.

Concluimos tras todo esto que aunque la precisión que se pueda obtener con funciones de $\mathcal{H}_n(X, Y)$ y $\mathcal{H}_n^+(X, Y)$ es diferente para un mismo número de neuronas n , añadiendo una más el sesgo es irrelevante y además $\mathcal{H}_n^+(X, Y)$ tiene mayor coste computacional, por lo que afirmamos que es un artificio de las redes neuronales multicapa para enlazar una capa con otra y que en redes neuronales de una capa oculta carece de sentido.

5.1.2. *Modus operandi* ante problemas que requieran un dominio de salida discreto

Es usual en la literatura presentar las redes neuronales con la salida compuesta con una función θ , de tal manera que una red neuronal sea de la forma

$$h_k(x) = \theta_k \left(\sum_{i=1}^n \beta_{ik} \gamma_i \left(w_{0i} + \sum_{j=1}^d w_{ji} x_j \right) \right) \text{ para cada } k \in \{1, \dots, s\}. \quad (5.2)$$

El teorema universal de convergencia 4.1 nos asegura que dado un número lo suficientemente grande de neuronas tal composición no es necesaria. Sin embargo, a nivel práctico ese número de neuronas puede no alcanzarse y, por cómo se produce la convergencia, el resultado serán funciones con imágenes con forma de dominios contenidos en \mathbb{R}^n .

Por la naturaleza de la imagen de ciertos problemas no sería necesaria mayor modificación y con nuestra definición sería más que suficiente. Pero en caso de que la salida tenga que cumplir alguna restricción, como por ejemplo en problemas de clasificación que necesiten de una salida discreta; sería necesario la composición con una función θ que discretice la imagen como observábamos en 4.7.

La situación por tanto es la siguiente, si procedemos como en 5.2, se tendría que si f es el patrón subyacente de la clasificación y θ la función que discretiza el dominio se deberá de encontrar $h \in \mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$ tal que h aproxime a $\theta^{-1} \circ f$.

Lo cual exige que θ tenga inversa y que sea medible, además de la cuestión de qué θ es la más conveniente ...

Aportación original

Es por ello que en nuestra aproximación descartamos este modo de proceder y proponemos el siguiente:

1. Entrenamos $h \in \mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$, esto dará una salida que no necesariamente será discreta.
2. Una vez que se ha calculado h , θ se determina a partir de h .

Construcción de θ

Una vez fijada h , para $A \subseteq \mathcal{D}$ se define el conjunto

$$\Lambda = \{(h(x), y) : (x, y) \in A\}.$$

A partir del cual θ será una función que se ha obtenido por el método de clasificación supervisada de los k -vecinos más cercanos (k -NN) donde el conjunto de entrenamiento es Λ .

El motivo por el que se ha optado por esta aproximación es que si consideráramos θ como una función a trozos usual definida como

$$\tilde{\theta}(x) = y \text{ si } h(x) \in I_y.$$

Donde I_y puede no ser un intervalo conexo y que introduciría complejidad al cálculo y determinación de $\tilde{\theta}$, es por ello que lo natural sería entonces construir un función que discretice a partir de los valores conocidos más cercanos, esto es el algoritmo k -NN.

Construcción de k -NN

Si la dimensión de salida con la que se ha entrenado h es uno, es decir $s = 1$. Por ser \mathbb{R} de una dimensión lo natural sería hacer $k = 1$, es decir

$$\theta(h(x)) = y_{\text{cercano}},$$

$$\text{donde } (h(a), y_{\text{cercano}}) \in A \text{ y } h(a) = \min\{|h(x) - h(b)| : (h(b), y) \in A\}.$$

La implementación de la fórmula anterior sería la siguiente:

Algoritmo 1: Cálculo de la función θ como clasificador 1-NN

- 1 **Input:** Conjunto de datos Λ .
 - 2 **Output:** Función $\theta : \mathbb{R} \rightarrow \mathbb{R}$
 - 1: {Definimos θ como}
 - Function** $\theta(h_x)$:
 - 2: $distanciaMnima \leftarrow \infty$
 - 3: $clase \leftarrow NADA$
 - 4: {Encontramos el más cercano}
 - para todo** $cada (h, y)$ **hacer**
 - ┌ **si** $distancia(h, h_x) < distanciaMnima$ **entonces**
 - └ $clase \leftarrow y$
 - └ **devolver** $clase$
 - └ **devolver** θ
-

5. Construcción técnica de las redes neuronales de una sola capa

5.1.3. Qué función de activación seleccionar

Los aspectos a tener en cuenta a la hora de seleccionar una función de activación frente a otra de una red neuronal serían los siguientes:

1. Espacio de memoria.
2. Coste computacional.
3. Efectividad en cuanto a reducir el error de aproximación.

Sobre la primera consideración está claro que el uso de una única función ahorraría el tener que almacenar el tipo de función que se va emplear en cada neurona.

Respecto al coste computacional puede uno basarse en un análisis teórico del número de operaciones y su complejidad o realizar un estudio empírico se ha realizado en la sección

6.1.2.

Fijado cierto número de neuronas, en lo que respecta a la precisión que nos puedan aportar las funciones de activación; por la idea intuitiva del funcionamiento de las funciones de activación mostrado en la sección 3.3 es un factor que repercute en los resultados pero desconocido a priori.

❖ Estrategia de selección de funciones de activación.

Esta idea abre la puerta a determinar mediante algoritmos genéticos qué funciones de activación podrían resultar más beneficiosas.

5.2. Construcción explícita y evaluación de una red neuronal

Ante todas las consideraciones expuestas y puesto que no existe ningún resultado o hipótesis a favor de combinar funciones de activación, en pos de simplificar el estudio, vamos a suponer a priori que todos los nodos están compuestos con la misma. Entonces, una red neuronal para nosotros vendrá determinada por dos matrices de pesos y una función de evaluación.

Es decir siguiendo la idea constructiva expuesta en manuales tales como [CM07] cualquier $h \in \mathcal{H}_n(\mathbb{R}^d, \mathbb{R}^s)$ de n neuronas en la capa oculta se implementa como $(\gamma, M_{n \times (1+r)}, M_{(n+1) \times s})$ con $M_{f \times c}$ matrices de dimensiones f filas y c columnas.

γ representa a función de activación en cada nodo. $M_{n \times (1+r)}$ son los pesos de la primera capa y $M_{(n+1) \times s}$ son los pesos de la salida.

Tomando como $\mathcal{J}_{r+1} : \mathbb{R}^r \rightarrow M_{(r+1) \times 1}(\mathbb{R})$ a la aplicación inyección que a cada vector (x_1, \dots, x_d) le hace corresponder el vector columna $(1, x_1, \dots, x_d)^T$. Además entendemos para una función $\gamma : \mathbb{R} \rightarrow \mathbb{R}$ que la imagen de la composición de una matriz con γ es evaluar cada una de sus entradas por γ . Con esta representación evaluar una red neuronal consistiría en el producto matricial de

$$h(x) = M_{(n+1) \times s} \cdot \mathcal{J}_{n+1} \left(\gamma \left(M_{n \times (1+d)} \cdot \mathcal{J}_{d+1}(x) \right) \right).$$

El coste computacional de tal operación es

Primera optimización reformulando la implementación de una red neuronal

Sin embargo de acuerdo a nuestro modelo definido en 4.1, nuestra propuesta de representación es la siguiente:

$$(\gamma, A, S, B) \text{ donde } A \in M_{n \times d}(\mathbb{R}), S \in M_{n \times 1}(\mathbb{R}) \text{ y } B \in M_{s \times n}(\mathbb{R}).$$

y con esta representación la evaluación sería

5.2. Construcción explícita y evaluación de una red neuronal

Operación	Apariciones
+	$nd + ns = n(d + s)$
×	$n(d + 1) + (n + 1)s = n(d + s + 2)$
γ	n
\mathfrak{J}	2

Tabla 5.1.: Coste computacional de la implementación propuesta en [AMMIL12]

$$h(x) = B \cdot \gamma (A \cdot x + S),$$

que tiene un coste computacional de

Operación	Apariciones
+	$nd + n(s - 1) = n(d + s - 1)$
×	$nd + ns = n(d + s)$
γ	n
\mathfrak{J}	0

Tabla 5.2.: Coste computacional de nuestra implementación de red neuronal de una capa

Que como vemos ha supuesto una mejora de

Operación	Coste formulación usual 5.1	Coste nuestra propuesta 5.2	Operaciones reducidas
+	$n(d + s)$	$n(d + s - 1)$	n
×	$n(d + s + 2)$	$n(d + s)$	$2n$
γ	n	n	0
\mathfrak{J}	2	0	2

Tabla 5.3.: Comparativas de coste computacional entre la implementación usual de una red neuronal y la nuestra

Además se ha reducido el espacio de memoria en n unidades del tipo de datos que utilicen las redes neuronales.

Ejemplo de evaluación de una red neuronal

A continuación vamos a dar un ejemplo basado en la red neuronal mostrada en la imagen 5.2, La red neuronal viene dada por las matrices: la matriz de la primera capa asociada es

$$A = \begin{bmatrix} 0.1 & -0.2 \\ -0.3 & -0.1 \end{bmatrix}$$

la matriz de sesgos

$$S = \begin{bmatrix} 0.3 \\ 0.5 \end{bmatrix}$$

La matriz de pesos de la capa oculta sería

$$B = \begin{bmatrix} 0.1 & -0.6 \\ -0.6 & 0.5 \end{bmatrix}.$$

5. Construcción técnica de las redes neuronales de una sola capa

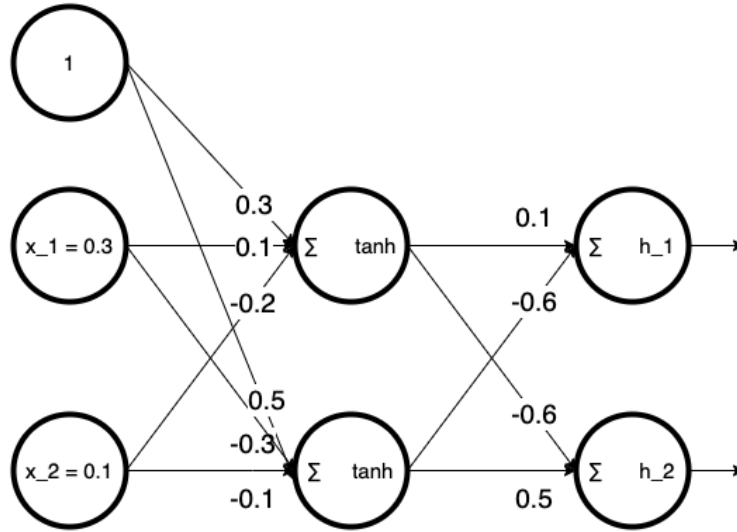


Figura 5.2.: Ejemplo de evaluación de una red neuronal de una capa con entrada y salida de tamaño dos y dos neuronas en la capa oculta

Para la entrada $x = (0.3, 0.1)$ los sucesivos cálculos serían

$$Ax = \begin{bmatrix} 0.1 & -0.2 \\ -0.3 & -0.1 \end{bmatrix} \begin{bmatrix} 0.3 \\ 0.1 \end{bmatrix} = \begin{bmatrix} 0.01 \\ -0.1 \end{bmatrix}.$$

$$Ax + S = \begin{bmatrix} 0.01 \\ -0.1 \end{bmatrix} + \begin{bmatrix} 0.3 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 0.31 \\ 0.4 \end{bmatrix}.$$

$$\gamma(Ax + S) = \gamma\left(\begin{bmatrix} 0.31 \\ 0.4 \end{bmatrix}\right) = \begin{bmatrix} \gamma(0.31) \\ \gamma(0.4) \end{bmatrix} = \begin{bmatrix} 0.3004 \\ 0.3799 \end{bmatrix}.$$

Finalmente las respectivas salidas de las capas ocultas serían

$$B\gamma(Ax + S) = \begin{bmatrix} 0.1 & -0.6 \\ -0.6 & 0.5 \end{bmatrix} \begin{bmatrix} 0.3004 \\ 0.3799 \end{bmatrix} = \begin{bmatrix} -0.1979 \\ 0.0097 \end{bmatrix}.$$

5.2.1. Implementación de una red neuronal y evaluación

Como conclusión a todo lo explicado una red neuronal no sería más que una estructura que almacenara (γ, A, S, B) , esto es

Algoritmo 2: Estructura de una red neuronal

```

1  Entrada:
    ■  $d$ : Dimensión de los datos de entrada.
    ■  $s$ : Dimensión que define una red neuronal.
    ■  $n$ : Número de nodos en la capa oculta.

    struct {
    |  $A \leftarrow$  Matriz  $n$  filas  $d$  columnas
    |  $S \leftarrow$  Matriz  $n$  filas 1 columnas
    |  $B \leftarrow$  Matriz  $s$  filas  $n$  columnas
    } Red neuronal( $d,s,n$ )

```

y si quisiéramos evaluar la red neuronal el algoritmo sería el siguiente

Algoritmo 3: Evaluación de una red neuronal, *Forward propagation*

```

1  Entrada:
    ■  $x$ : Vector de atributos que se desea evaluar con la red neuronal.
    ■  $h = (\gamma, A, S, B)$  representa la red neuronal que se desea evaluar y la función de
      activación  $\gamma$  con la que se va a evaluar.

    Function ForwardPropagation( $h, x$ ):
    |  $sensibilidad \leftarrow A \cdot x + S$  ;
    |  $primeraCapa \leftarrow$  Se crea vector del mismo tamaño que  $sensibilidad$  ;
    | para cada  $entrada\text{-}l\acute{e}sima\text{-}Neurona \in sensibilidad$  hacer
    | |  $primeraCapa[i] \leftarrow \gamma( entrada\text{-}l\acute{e}sima\text{-}Neurona )$ 
    | ;
    |  $salida \leftarrow B \cdot primeraCapa$  ;
    | devolver  $salida$  ;

```

5.3. Aprendizaje

Se entiende por aprendizaje de una red neuronal como el proceso por el cual se determina el valor sus pesos, es decir, lo que en el ejemplo 5.2 consistía en las matrices A , S y B .

5.3.1. Método de gradiente descendente y *backpropagation*

De acorde a los capítulos uno y dos del libro [CM07], una vez concretado el problema y sus elementos (2.1.1) es necesario definir un método con el que aproximar la función ideal f , para ello introduciremos el algoritmo de gradiente descendente. Se trata de un método iterativo de minimización de funciones diferenciables.

En nuestro caso particular se quiere aproximar la función ideal desconocida f a partir de funciones (redes neuronales) $h \in \mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$, concretamente se fijará un número n de neuronas en la capa oculta. Dada también una función de error diferenciable y que no presente puntos de inflexión $E : \mathcal{H}_n(\mathbb{R}^d, \mathbb{R}^s) \rightarrow \mathbb{R}$, se toma red neuronal cualquiera $h_0 \in \mathcal{H}_n(\mathbb{R}^d, \mathbb{R}^s)$ y fijado $\eta \in \mathbb{R}^+$.

Se define la sucesión

$$h_{t+1} = h_t - \eta \nabla E(h_t). \tag{5.3}$$

Donde h_n es una sucesión cuyos términos convergen a un mínimo local.

Observaciones sobre el algoritmo

- El algoritmo solo encuentra óptimos locales con una dependencia crucial del valor de inicio.
- La convergencia no es segura en un tiempo finito y requiere de criterios de parada.
- Debe de fijarse el número de neuronas en la capa oculta a priori.
- Si la función es convexa el mínimo será global.
- El parámetro η puede ser cualquiera y debe de ser fijado o controlado por el diseñador.

Con el fin de reducir el coste del cálculo del gradiente, se utiliza generalmente el algoritmo conocido como *backpropagation* que fue publicado en 1989 en el artículo [RHW86]. Las hipótesis bajo las que fue diseñado tal algoritmo fueron: redes neuronales con varias capas y entendiéndose por red neuronal un modelo más costoso al nuestro. **Es por ello que a continuación plantearemos una versión propia deducida y optimizada de acorde a nuestro modelo.**

Sea $E_{in} : \mathcal{H}(\mathbb{R}^d, \mathbb{R}^s) \rightarrow \mathbb{R}_0^+$ la función para medir error habitualmente usada, la cual tomaremos como el error dentro del conjunto de entrenamiento, esto es, si el conjunto de entrenamiento \mathcal{D} está constituido por N datos de la forma (x_i, y_i) con x_i el vector de entrada o atributos y y_i el estado o valor deseado para cualquier $k \in \{1, \dots, N\}$. Para cualquier $h \in \mathcal{H}(\mathbb{R}^d, \mathbb{R}^s)$ denotando por h_k a su k -ésima proyección, se define como métrica de error dentro de \mathcal{D} como

$$E_{in}(h) = \frac{1}{N} \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k)^2.$$

puesto que $\frac{1}{N}$ no es más que una constante de proporcionalidad (y que por tanto no afecta a la minimización) del error y que además puede ser corregida en 5.3 con η , con el fin de

Ⓢ Aclaración gradiente red neuronal

Puede a priori uno confundirse con la notación, pero recordemos que las redes neuronales estaban determinadas por sus parámetros (matrices), luego lo único que se está haciendo es derivar con respecto a tales parámetros.

Ⓢ Idea general del algoritmo gradiente descendente

Cada iteración se obtendrá una nueva red neuronal con un error menor dentro de los datos de entrenamiento del conjunto.

⚡ Consecuencia del requisito de diferenciabilidad de $E(h)$ $E(h)$ será diferenciable si y sólo si las funciones de activación lo son.

ahorrar coste computacional la fijaremos a conveniencia. Es decir, podemos suponer que nuestra función de error a minimizar es

$$E_{in}(h) = \frac{1}{2} \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k)^2.$$

Antes de adentrarnos en los cálculos tengamos presente que hemos definido una red neuronal $h \in \mathcal{H}_n(\mathbb{R}^d, \mathbb{R}^s)$ como $h = (h_1, \dots, h_s)$ con cada proyección k -ésima dada por:

$$h_k(x) = \sum_{j=1}^n \beta_{jk} \sigma \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \quad (5.4)$$

para la cual hemos impuesto que la función de activación σ sea diferenciable.

Denotaremos como $\chi_{[c]}$ a la función característica:

$$\chi_{[c]} = \begin{cases} 1 & \text{si se satisface } c \\ 0 & \text{si no se satisface } c. \end{cases}$$

Así pues, en base al modelo expuesto en 5.4 a la hora de calcular el gradiente del error con respecto a los parámetros que determinan la red, tendríamos tres tipos de derivadas parciales, las dependientes de β_{jk} , las de α_{0j} y las de α_{ij} , para cada caso concreto y en virtud de la regla de la cadena, se tiene:

- Derivada parcial del error con respecto a β_{vw} donde $v \in \{1, \dots, n\}$ y $w \in \{1, \dots, s\}$:

$$\begin{aligned} \frac{\partial E(h)}{\partial \beta_{vw}} &= \frac{\partial}{\partial \beta_{vw}} \left[\frac{1}{2} \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k)^2 \right] & (5.5) \\ &= \frac{1}{2} \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s 2 (h_k(x) - y_k) \frac{\partial h_k(x)}{\partial \beta_{vw}} \\ &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \frac{\partial}{\partial \beta_{vw}} \left[\sum_{j=1}^n \beta_{jk} \sigma \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \right] \\ &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\sum_{j=1}^n \frac{\partial}{\partial \beta_{vw}} \left[\beta_{jk} \sigma \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \right] \right) \\ &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\sum_{j=1}^n \chi_{[j=v \wedge k=w]} \sigma \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \right) \\ &= \sum_{(x,y) \in \mathcal{D}} (h_w(x) - y_w) \left(\sigma \left(\alpha_{0v} + \sum_{i=1}^d \alpha_{iv} x_i \right) \right). \end{aligned}$$

5. Construcción técnica de las redes neuronales de una sola capa

- Derivada parcial del error con respecto a α_{0v} donde $v \in \{1, \dots, n\}$:

$$\begin{aligned}
 \frac{\partial E(h)}{\partial \alpha_{0v}} &= \frac{\partial}{\partial \alpha_{0v}} \left[\frac{1}{2} \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y)^2 \right] & (5.6) \\
 &= \frac{1}{2} \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s 2(h_k(x) - y_k) \frac{\partial h_k(x)}{\partial \alpha_{0v}} \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \frac{\partial}{\partial \alpha_{0v}} \left[\sum_{j=1}^n \beta_{jk} \sigma \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \right] \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\sum_{j=1}^n \beta_{jk} \frac{\partial}{\partial \alpha_{0v}} \left[\sigma \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \right] \right) \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\sum_{j=1}^n \beta_{jk} \sigma' \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \frac{\partial}{\partial \alpha_{0v}} \left[\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right] \right) \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\sum_{j=1}^n \beta_{jk} \sigma' \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \chi_{[j=v]} \right) \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\beta_{vk} \sigma' \left(\alpha_{0v} + \sum_{i=1}^d \alpha_{iv} x_i \right) \right).
 \end{aligned}$$

- Derivada parcial del error con respecto a α_{uv} donde $u \in \{1, \dots, d\}$ y $v \in \{1, \dots, n\}$:

$$\begin{aligned}
 \frac{\partial E(h)}{\partial \alpha_{uv}} &= \frac{\partial}{\partial \alpha_{uv}} \left[\frac{1}{2} \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k)^2 \right] & (5.7) \\
 &= \frac{1}{2} \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s 2(h_k(x) - y_k) \frac{\partial h_k(x)}{\partial \alpha_{uv}} \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \frac{\partial}{\partial \alpha_{uv}} \left[\sum_{j=1}^n \beta_{jk} \sigma \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \right] \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\sum_{j=1}^n \beta_{jk} \frac{\partial}{\partial \alpha_{uv}} \left[\sigma \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \right] \right) \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\sum_{j=1}^n \beta_{jk} \sigma' \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \frac{\partial}{\partial \alpha_{uv}} \left[\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right] \right) \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\sum_{j=1}^n \beta_{jk} \sigma' \left(\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i \right) \chi_{[i=u \wedge j=v]} x_i \right) \\
 &= \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\beta_{vk} \sigma' \left(\alpha_{0v} + \sum_{i=1}^d \alpha_{iv} x_i \right) x_u \right).
 \end{aligned}$$

Recapitulando los resultados, el cálculo de las derivadas parciales con $u \in \{1, \dots, d\}$, $v \in \{1, \dots, n\}$ y $w \in \{1, \dots, s\}$ consiste en:

- Por el desarrollo (5.5) la derivada parcial del error con respecto a β_{vw} es:

$$\frac{\partial E(h)}{\partial \beta_{vw}} = \sum_{(x,y) \in \mathcal{D}} (h_w(x) - y_w) \left(\sigma \left(\alpha_{0v} + \sum_{i=1}^d \alpha_{iv} x_i \right) \right).$$

- Por el desarrollo (5.6) derivada parcial del error con respecto a α_{0v} es :

$$\frac{\partial E(h)}{\partial \alpha_{0v}} = \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\beta_{vk} \sigma' \left(\alpha_{0v} + \sum_{i=1}^d \alpha_{iv} x_i \right) \right).$$

- Por el desarrollo (5.7) derivada parcial del error con respecto a α_{uv} :

$$\frac{\partial E(h)}{\partial \alpha_{uv}} = \sum_{(x,y) \in \mathcal{D}} \sum_{k=1}^s (h_k(x) - y_k) \left(\beta_{vk} \sigma' \left(\alpha_{0v} + \sum_{i=1}^d \alpha_{iv} x_i \right) x_u \right).$$

Si el cálculo se hiciera sin tener más consideración alguna que la propia expresión e incluso obviando el coste de aplicar *forward propagation*, la diferencia y el producto supondría el cómputo que mostramos en la tabla 5.4.

	Número de parámetros	+ / -	× / ÷	σ	σ'
(5.5) $\frac{\partial h(x)}{\partial \beta_{ik}}$	ns	$ns(d+1)$	nsd	ns	o
(5.6) $\frac{\partial h(x)}{\partial \alpha_{0i}}$	n	nd	$n(d+1)$	o	n
(5.7) $\frac{\partial h(x)}{\partial \alpha_{ij}}$	nd	nd^2	$nd(d+2)$	o	nd
<i>forward propagation</i>	$n(s+1+d)$	$n^2(d+s)(s-1+d)$	$n^2(s+1+d)(d+s)$	n	o

Tabla 5.4.: Coste computacional de aplicar directamente el algoritmo de gradiente descendente para actualizar $h \in \mathcal{H}_n(\mathbb{R}^d, \mathbb{R}^s)$

Abordar el problema de manera directa es muy ineficiente. Basta fijar un par $(x, y) \in \mathcal{D}$ para darse cuenta de que el cálculo de $h(x) - y$ se repite $n(s+1+d)$ veces y que además, como mostramos en la tabla 5.5 hay cálculos que se repiten en (5.5), (5.6) y (5.7).

Expresión repetida en	$\frac{\partial E(h)}{\partial \beta_i}$	$\frac{\partial E(h)}{\partial \alpha_{0i}}$	$\frac{\partial E(h)}{\partial \alpha_{ji}}$	Total apariciones para i	Coste total	Coste en memoria
$\alpha_{0i} \sum_{j=1}^d \alpha_{ji} x_j$	1	1	d	$d+2$	$n(d+2)$	n
$\beta_{ik} \sigma' \left(\alpha_{0i} \sum_{j=1}^d \alpha_{ji} x_j \right)$	o	1	d	$d+1$	$ns(d+1)$	ns

Tabla 5.5.: Veces que se calcula una misma expresión para el cálculo de gradiente descendente fijado un i y coste en memoria en unidades si se almacenara el cálculo de todos esos parámetros.

Notemos además que las expresiones del tipo $\alpha_{0j} + \sum_{i=1}^d \alpha_{ij} x_i$ son las que denotábamos como *sensibilidades* en el algoritmo de *forward propagation* 3.

5.3.2. Motivación para almacenar cálculos parciales

A la vista de las repeticiones de cálculos, su coste computacional, ofrece un gran beneficio almacenar $h_k(x) - y_k$, ya que el coste de cálculo aumenta de manera lineal conforme aumenta el número de neuronas mientras que se mantiene constante el costo en memoria.

Para el resto de expresiones podría no estar tan claro su beneficio, ya que estamos en una situación en la que el coste en memoria y de cálculo aumentan de manera lineal con respecto al número de neuronas.

Sin embargo, para un cálculo directo sin almacenar operaciones intermedias, sería necesario reservar en memoria espacio para $n(s + 1 + d)$ parámetros, ya que no se debería de sobrescribir los parámetros antiguos con los nuevos ya que afectaría al cálculo.

Por otra parte, hemos observado que el coste en memoria de ir almacenando los resultados parciales 5.5 y el propio coste en memoria de nuestro algoritmo 5 supondrían un espacio de $n(d + 2s + 3) + 1$. Así pues se acaba de probar que un algoritmo directo es peor en coste de cálculo y ligeramente mejor en memoria frente a otro que almacene los resultados parciales.

Por lo que los algoritmos que proponemos a continuación se basarán en almacenar estos resultados.

5.3.3. Algoritmos de actualización de pesos de una neurona

Nuestro objetivo es aplicar el algoritmo de gradiente descendente, codificaremos una red neural $h \in \mathcal{H}_n(\mathbb{R}^d, \mathbb{R}^s)$ de estructura (A, S, B) , como dos matrices de parámetros α y β y de respectivos tamaños $d \times (n + 1)$ y $s \times n$:

$$A = (\alpha_{ij}) \text{ con } i \in \{1, \dots, d\}, j \in \{1, \dots, n\}.$$

$$S = (\alpha_{0j}) \text{ con } j \in \{1, \dots, n\}.$$

$$B = (\beta_{jk}) \text{ con } j \in \{1, \dots, n\}, k \in \{1, \dots, s\}.$$

De esta manera h modificará el valor de sus pesos de acorde al algoritmo de gradiente descendente 5.3 que viene dado por

Algoritmo 4: Algoritmo gradiente descendente conocidas las derivadas parciales.

- 1 **Input:** h red neuronal y conjunto de entrenamiento
 - 2 **Output:** h actualizada de acorde al algoritmo de gradiente descendente.
 - 1: Debe de calcularse el previamente $\nabla E(h)$, es decir cada una de las derivadas parciales, esto se hará en el algoritmo 5.
 - 2: Actualización de los pesos de A

para $j \in \{1, \dots, n\}$ **hacer**

para $i \in \{1, \dots, d\}$ **hacer**

$$\alpha_{ij} \leftarrow \alpha_{ij} - \eta \frac{\partial E(h)}{\partial \alpha_{ij}}$$
 - 3: Actualización de los pesos de S

para $j \in \{1, \dots, n\}$ **hacer**

$$\alpha_{0j} \leftarrow \alpha_{0j} - \eta \frac{\partial E(h)}{\partial \alpha_{0j}}$$
 - 4: Actualización de los pesos de B

para $k \in \{1, \dots, s\}$ **hacer**

para $j \in \{1, \dots, n\}$ **hacer**

$$\beta_{jk} \leftarrow \beta_{jk} - \eta \frac{\partial E(h)}{\partial \beta_{jk}}$$
-

Notemos la necesidad de una variable $\eta \in \mathbb{R}$ que es prefijada. En la terminología de aprendizaje automático se suele referir a este término como tasa de aprendizaje o *learning rate* una amplia literatura sobre cómo debe seleccionarse y variantes adaptativas del mismo, véase [AMMIL12].

La complejidad algorítmica del algoritmo de gradiente descendente recae principalmente en el coste de calcular los gradientes, la cual calculamos en la sección 2. En resumen su complejidad es

$$\begin{aligned} & \mathcal{O}(\max\{nd, ns, \text{coste cálculo de los gradientes}\}) = \\ & = \mathcal{O}(\text{coste cálculo de los gradientes}) = \\ & = \mathcal{O}(|\mathcal{D}|nds). \end{aligned}$$

Procederemos ahora a determinar el cálculo del gradiente, como hemos visto este viene determinado por la expresión.

El algoritmo que se muestra a continuación supone que la memoria no es un problema así que se plantea para que no haya ningún cálculo repetido.

5. Construcción técnica de las redes neuronales de una sola capa

Algoritmo 5: Algoritmo cálculo del gradiente $\nabla E(h)$.

```

1  Input:  $h \in \mathcal{H}_n(\mathbb{R}^d, \mathbb{R}^s)$  red neuronal que representaremos como en 5.3.3 y  $\mathcal{D}$ 
    conjunto de entrenamiento, con pares  $(x, y)$ .
2  Output: Gradiente del error, esto es  $\nabla E(h)$  que se almacena en las siguientes
    variables:
    ■ parcial $\alpha_{ij}$  hace referencia a  $\frac{\partial E(h)}{\partial \alpha_{ij}}$ .
    ■ parcial $\beta_{ij}$  hace referencia a  $\frac{\partial E(h)}{\partial \beta_{ij}}$ .

1: Inicializamos respectivamente las variable que contendrán a las derivadas parciales
    ■ parcial $\alpha_{ij} \leftarrow$  matriz de reales de tamaño  $n + 1 \times d$  con entradas a cero.
    ■ parcial $\beta_{ij} \leftarrow$  matriz de reales de tamaño  $s \times n$  con entradas a cero.

2: para cada para  $(x, y) \in \mathcal{D}$  hacer
3: Calculamos forward propagation para cada  $i \in \{1, \dots, n\}$  hacer
4: Cálculo de las sensibilidades vector  $\delta$  donde su componente  $i$ -ésima viene
    dada por
     $\delta_i \leftarrow \alpha_{0i} + \sum_{j=1}^d \alpha_{ji} x_j$ .
5: Cálculo de la salida de los nodos, vector  $s$  donde cada componente es
     $s_i \leftarrow \sigma(\delta_i)$ .
6: Almacenamos las derivadas
     $ds_i \leftarrow \sigma'(\delta_i)$ .
    para cada  $k \in \{1, \dots, s\}$  hacer
    └ derivada $\beta_{ik} \leftarrow \beta_{ik} ds_i$ 
7:  $h_x \leftarrow Bs$  (se multiplica la matriz de coeficientes de la red neuronal por la salida
    previamente calculada)
8: diferencia  $\leftarrow h_x - y$ 
9: Cálculo de gradiente de  $\frac{\partial E(h)}{\partial \beta_{uv}}$ 
    para  $u \in \{1, \dots, n\}$  hacer
    └ para  $v \in \{1, \dots, s\}$  hacer
        └ parcial $\beta_{uv} \leftarrow$  parcial $\beta_{uv} +$  diferencia $_v s_i$ .
10: Cálculo del tipo  $\frac{\partial E(h)}{\partial \alpha_{uv}}$ 
    para  $v \in \{1, \dots, n\}$  hacer
    └ para  $k \in \{1, \dots, s\}$  hacer
        └ auxiliarDiferenciaPorDerivada  $\leftarrow$  diferencia $_k$  derivada $\beta_{vk}$ .
        └ parcial $\alpha_{0v} \leftarrow$  parcial $\alpha_{0v} +$  auxiliarDiferenciaPorDerivada.
        └ para  $u \in \{1, \dots, d\}$  hacer
            └ parcial $\alpha_{uv} \leftarrow$  parcial $\alpha_{uv} +$  auxiliarDiferenciaPorDerivada  $x_u$ .

```

Análisis de la complejidad

Notemos que la complejidad computacional del algoritmo 5 es:

- **Coste de cálculo:** Entendiendo suma, productos y evaluaciones como constante de

valor 1 el resultado es:

$$\begin{aligned}
 \text{coste} &= |\mathcal{D}| (n(1 + 1 + 1 + s) + 1 + 1 + ns + ns(1 + 1 + d)) \\
 &= |\mathcal{D}| (n(3 + s) + 3ns + nsd) \\
 &= |\mathcal{D}| (4ns + dns + 3n) \\
 &\leq |\mathcal{D}| 3 \max\{4, d\} ns.
 \end{aligned}$$

La complejidad del algoritmo resulta por tanto $\mathcal{O}(|\mathcal{D}|nds)$. De aquí se deduce, puesto que s, d son constantes fijas que a nivel práctico serán mucho menor que el número de neuronas n y el número de datos de entrenamiento $|\mathcal{D}|$, que lo que afecta primordialmente al coste es $|\mathcal{D}|$ y n .

Si queremos converger con cierta precisión, el teorema de aproximación no nos permite reducir n . Sin embargo, si suponemos que \mathcal{D} presenta datos independientes e idénticamente distribuidos tomar un subconjunto aleatorio mantendría un buen estimador del error actual y por ende del aprendizaje.

La clave entonces reside en qué tamaño de subconjunto tomar, en artículos como [PKK⁺19] se ofrece una posible estimación basada en reducir el número de épocas, es decir ejecuciones del algoritmo de gradiente descendente.

- **Coste en memoria:** El coste total de memoria, entendiendo por una unidad aquella que almacene un parámetro concreto de la red neuronal será:

$$\begin{aligned}
 \text{Coste en memoria} &= nd + n(s + 1) && \text{parciales} \\
 &+ n(1 + 1 + s) && \text{cálculo de } \delta, s, ds \text{ y derivada } \beta \\
 &+ 1 && \text{variable auxiliar} \\
 &= n(d + 2s + 3) + 1.
 \end{aligned}$$

5.3.4. Otras alternativas al alternativo de gradiente descendente

Recordemos que nuestro enfoque partía de fundamentar toda decisión de diseño de manera rigurosa. Destaquemos por tanto que en el algoritmo de gradiente descendente se introduce la restricción de que las funciones de activación deben de ser diferenciables.

Esto, a priori, no es de extrañar, ya que si buscamos optimizar algún aspecto, en algún momento deberemos de particularizar o reenfocar algunos de los componentes del problema, sin embargo ¿exigir tales restricciones está sustentado teóricamente?

Desde un punto de vista analítico la forma por excelencia de minimizar radica en la derivación. Pudiéndose relajar el concepto de derivada a derivada débil o incluso trabajar en el ambiente de teoría de distribuciones, donde *casi todo* se puede derivar [Lig58]. La clave sería poder implementar los cálculos y obtener buenos resultados experimentales.

La cuestión reside entonces si existen otras alternativas al algoritmo de gradiente descendente podrían reducir el costo computacional del proceso de aprendizaje.

Para ello se ha consultado el estado del arte actual encontrando publicaciones como:

5.3.4.1. Gradients without Backpropagations

A finales de este mismo años, se publicó el artículo *Gradients without Backpropagations* [BPS⁺22], en él se introduce el algoritmo al que acuñan como *forward propagation* y al que posicionan los propios autores como una alternativa *más eficiente en coste* que el algoritmo de *Backpropagation*.

5. Construcción técnica de las redes neuronales de una sola capa

Por desgracia, en este artículo no se da una demostración formal que verdaderamente explique el beneficio computacional, sino que se basan en meras experimentaciones.

Sin embargo, tiene su interés y es por ello que lo mencionamos, en indicar que existe margen de mejora imponiendo como restricción el que las funciones de activación sean diferenciables. Esta tendencia se puede ver también en artículos como [WY10].

5.4. Inicialización de los pesos de una red neuronal

Como observábamos en la sección 5.3.1, el gradiente descendente pretende en cada iteración mejorar la solución encontrada, pero es totalmente sensible a la posición inicial de los pesos. Presentamos por tanto la siguiente propuesta para inicializar una red neuronal a modo de intentar que sus pesos se encuentren ya cerca de la solución. Destaquemos que este algoritmo no solo servirá exclusivamente para el método de gradiente descendente sino para cualquier otro dependiente del punto inicial.

5.4.1. Estado del arte relacionado

Se ha comprobado en la práctica que el uso de *backbones* reporta mejoras sustanciales en el entrenamiento de redes neuronales, véanse artículos como [LCL⁺21], [GCZ⁺19] y sus referencias.

Cuando se carecen de *backbones* o no es posible utilizarlos, una buena técnica para dar un valor inicial a los pesos consiste en basarse en metaheurísticas tales como algoritmos genéticos, ver [Mono2], sin embargo estos tienen un coste computacional importante.

Ante esta situación, nuestro algoritmo propone una inicialización de bajo coste computacional y que no necesita de más valores que los del conjunto de entrenamiento.

5.4.2. Descripción del método propuesto

Aportación original

La idea proviene de la demostración casi constructiva del teorema 4.8.

Se desea inicializar los pesos de $h \in \mathcal{H}_n(\mathbb{R}^d, \mathbb{R}^s)$, para la cual, una vez fijado el número n de neuronas de nuestra red neuronal, será necesario determinar un subconjunto $\Lambda \mathcal{D}$ de datos de entrenamiento.

La bondad del resultado depende en gran medida de Λ , puesto que a priori se carece de hipótesis, se seleccionará de manera aleatoria bajo supuesto de una distribución independiente e idénticamente distribuida de los datos.

Como apunta la demostración, debe de encontrarse un $p \in \mathbb{R}^d$ satisfaciendo que $p \cdot (x_i - x_j) \neq 0$ para cualesquiera atributos x_i, x_j distintos de Λ .

Es decir que se estaría considerando un vector que no pertenezca a una unión finita de hiperplanos ortogonales de \mathbb{R}^d . De manera teórica la probabilidad de seleccionar un p y que pertenezca al espacio ortogonal es 0, sin embargo esto no quiere decir que no pueda pasar.

Tomaremos por tanto un p aleatorio y a partir de él seleccionaremos Λ lo suficientemente grande para que al menos n vectores admitan de manera estricta la ordenación:

② **Qué es un backbone**
Se denomina *backbone* a los nodos de una red neuronal que han sido **inicializados con valores de otras redes neuronales ya entrenadas**. Por ejemplo si quisiéramos clasificar insectos y partiéramos de una red neuronal que se entrenó para clasificar frutas.

$$p \cdot x_1 < p \cdot x_2 < \dots < p \cdot x_n. \quad (5.8)$$

Para continuar, para la función de activación seleccionada γ , por cómo se definen existirá un $M \in \mathbb{R}^+$ tal que

$$\gamma(K) = 1 \text{ y } \gamma(-K) = 0 \text{ sean constantes para todo } K \geq M. \quad (5.9)$$

Una vez concretados los valores p , Λ y M que satisfagan las condiciones (5.8) y (5.9) falta concretar los valores iniciales de la red neuronal.

Para ello debemos de calcular el valor de las matrices (A, S, B) que definen a una red neuronal y que presentamos en la sección 5.2.1.

Recordemos que A y S tienen tantas filas como neuronas y B tantas columnas como neuronas.

Usado la notación vectorial $p_{[i,j]} = (p_i, p_{i+1}, \dots, p_j)$ donde $(p_0, p_1, \dots, p_d) = p$, comenzaremos definiendo el valor de la primera fila como

$$\begin{aligned} S_1 &= Mp_0, \\ A_{1*} &= Mp_{[1,d]}, \\ B_{*1} &= y_1. \end{aligned}$$

Los valores de la fila k -ésimas de las matrices (A, S) , vendrán determinados por la única función afín $A \in \mathcal{A}(\mathbb{R}^d)$, dada por $A_k(x) = B_k(p \cdot x)$, con B_k como la única función afín de \mathbb{R} en \mathbb{R} que cumple que

$$B_k(p \cdot x_{k-1}) = -M \text{ y } B_k(p \cdot x_k) = M.$$

Que esto equivale a calcular las constantes reales $\tilde{\alpha}$.

Si tenemos presente que

$$\tilde{\alpha}_{kp}(p \cdot x_{k-1}) + \tilde{\alpha}_{ks} = -M \text{ y } \tilde{\alpha}_{kp}(p \cdot x_k) + \tilde{\alpha}_{ks} = M.$$

Resolviendo el sistema resulta que

$$\begin{cases} \tilde{\alpha}_{kp} = \frac{2M}{p \cdot (x_k - x_{k-1})} \\ \tilde{\alpha}_{ks} = M - \tilde{\alpha}_{kp}(p \cdot x_{k-1}) = M - \frac{2M}{p \cdot (x_k - x_{k-1})}(p \cdot x_{k-1}) \end{cases}$$

Luego los coeficientes de la red neuronal A, S se deduciría de

$$\begin{cases} \alpha_{k0} = \tilde{\alpha}_{ks} = M - \frac{2M}{p \cdot (x_k - x_{k-1})}(p \cdot x_{k-1}) \\ \alpha_{ki} = \tilde{\alpha}_{kp} p_i = \frac{2M}{p \cdot (x_k - x_{k-1})} p_i \end{cases}$$

Esto define un sistema lineal compatible cuya solución son las respectivas filas y columnas:

5. Construcción técnica de las redes neuronales de una sola capa

$$\begin{cases} S_k = M - \frac{2M}{p \cdot (x_k - x_{k-1})} (p \cdot x_{k-1}) \\ A_{ki} = \frac{2M}{p \cdot (x_k - x_{k-1})} p_i \\ B_{*k} = y_k - y_{k-1} \end{cases}$$

Con todo esto el proceso algorítmico resultante es:

Algoritmo 6: Inicialización de pesos de una red neuronal

- 1 **Input:** Tamaño red neuronal n , conjunto de datos de entrenamiento \mathcal{D} , constate M involucrada en 5.9.
- 2 **Input:** Red neuronal, representada con las matrices (A, S, B) .

1: Inicializamos p .

$p \leftarrow$ vector de \mathbb{R}^{d+1} . {Como heurística será generado con distribución uniforme en el intervalo $[0, 1]$ }

2: Selección de los datos de inicialización $\Lambda \subset \mathcal{D}$.

$$\Lambda \leftarrow \{\emptyset\}$$

mientras tamaño de $\Lambda < n$ **hacer**

Tomamos de manera aleatoria (x, y) de \mathcal{D} .

si para todo $(a, b) \in \Lambda$ se satisface que $p \cdot (x - a)$ **entonces**

$$\Lambda \leftarrow \Lambda \cup \{(x, y)\}.$$

{ Λ está ordenado conforme a la propiedad 5.8 }

3: Cálculo de los parámetros base de la red neuronal.

Para el primer $(x_1, y_1) \in \Lambda$

$$S_1 = Mp_0,$$

$$A_{1*} = Mp_{[1,d]},$$

$$B_{*1} = y_1.$$

$$\Lambda \leftarrow \Lambda \setminus \{(x_1, y_1)\}$$

4: Cálculo del resto de neuronas. **para** cada $(x_k, y_k) \in \Lambda$ **hacer**

$$S_k = M - \frac{2M}{p \cdot (x_k - x_{k-1})} (p \cdot x_{k-1})$$

$$A_{ki} = \frac{2M}{p \cdot (x_k - x_{k-1})} p_i \quad i \in \{1, \dots, d\},$$

$$B_{*k} = y_k - y_{k-1}.$$

5: **return** (A, S, B) .

5.4.2.1. Coste computacional algoritmo de inicialización de pesos

Antes de comenzar con el análisis notemos que el algoritmo tiene un componente aleatorio introducido en la selección del p ; y que en el peor (y con probabilidad nula) de los casos

podría acabar con coste $\mathcal{O}(|\mathcal{D}|)$ devolviendo un resultado erróneo.

Sin embargo, en virtud de las observaciones hechas en la descripción del método, la mayoría de los datos del conjunto de entrenamiento tomados para inicializar la red neuronal cumplirán la propiedad de ortogonalidad y por tanto una buena heurística es suponer que el paso 2 del algoritmo, *Selección de los datos de inicialización* se va a realizar con un coste de orden lineal a n .

De esta manera el algoritmo de inicialización de pesos propuesto tendría una complejidad $\mathcal{O}(n)$.

Notemos que este coste es bastante menor al de realizar *backpropagation* y que además éste debería de realizarse repetidamente para mejorar el error considerablemente, mientras que el nuestro tan solo una vez.

5.4.2.2. Observaciones

Observemos nuestro algoritmo, para un mismo conjunto de entrenamiento es capaz de producir infinitas soluciones diferentes ya que existen dos variables libres M y p .

La constante M depende de la función de activación seleccionada y recordemos que debe de escogerse para que satisfaga la condición 5.9.

Presentamos a continuación en la tabla 5.6 el valor M para alguna función de activación.

Función de activación	Valor mínimo de M
Función rampa	1
<i>Cosine Squasher</i>	$\frac{\pi}{2}$
Función indicadora o	0

Tabla 5.6.: Valor mínimo del parámetro M en algoritmo de inicialización de redes neuronales según la función de activación seleccionada.

5.4.2.3. Generalización del método para funciones de activación

A priori si la función de activación no cumple las propiedades demandadas no podría ser utilizada en el algoritmo. Sin embargo, es posible ver que se puede generalizar para funciones de activación más generales como definidas en 6.1.

- Por el teorema 6.1 también será válido para **funciones de activación que sean afines a una que satisfice 5.9**. El proceso constructivo consistiría en: (1) hacer que la red aprenda con la función que cumple los requisitos 5.9. (2) Los pesos obtenidos transformarlos con la misma técnica que se aplica en la demostración del teorema 6.1.
- **Funciones de activación asintóticas a 0 o 1**, esto es funciones que satisfacen que:
 1. $\lim_{x \rightarrow \infty} \psi(x) = 1$.
 2. $\lim_{x \rightarrow -\infty} \psi(x) = 0$.
 3. Que cumpla que $\psi(x) \neq 1$ para todo $x \in \mathbb{R}$ o $\psi(x) \neq 0$ para todo $x \in \mathbb{R}$.

Bastará con tomar un M lo suficientemente grande.

- Para funciones del tipo anterior, pero asintóticas a $a, b \in \mathbb{R}$, con alguno de los extremos a, b distintos de 0 y 1, bastará con realizar una transformación afín f cumpliendo que $f(a) = 0$ y $f(b) = 1$ y aplicar teorema 6.1.

5. Construcción técnica de las redes neuronales de una sola capa

De esta manera se pueden ampliar las funciones de activación válidas, añadiendo algunas como:

Función de activación	Valor mínimo de M
Función rampa	1
<i>Cosine Squasher</i>	$\frac{\pi}{2}$
Función indicadora o	0
Función de activación	Valor mínimo de M
Sigmoidea	con $M = 10$ el error menor de 10^{-5}
Tangente hiperbólica	con $M = 7$ el error menor de 10^{-5}
<i>Hardtanh</i>	1

Tabla 5.7.: Valor mínimo del parámetro M en algoritmo de inicialización de redes neuronales según la función de activación seleccionada (con más resultados).

TODO issue 107: Hacer experimentaciones

Parte II.

Estudio experimental y exploración hipótesis planteadas

6. Funciones de activación

Como se observó en 4.2 se desconoce teóricamente si una función de activación va a ser mejor que otra, es por ello y puesto que nuestro objetivo es reducir el coste computacional de una red neuronal que procederemos a realizar un análisis del costo de las redes neuronales.

6.1. Caracterización de las funciones de activación

Si bien por conveniencia teórica definimos las funciones de activación en 4.2 como una función de $\phi : \mathbb{R} \rightarrow [0, 1]$, no decreciente, con uno como límite en infinito y cero como límite a menos infinito.

Nos basaremos en el concepto *actual* de función de activación: basta con que sea una función no polinómica (véanse los artículos [SM15], [ADIP20] y [Fun89]), además también incluiremos la función de activación identidad.

Funciones no polinómicas hay infinitas y reincidentes en que a priori no hay una mejor que otra; por tanto, como criterio de selección nos guiaremos por la intuición que nos brinda la demostración del teorema 4.4. La imagen de una función de activación es relevante a la hora de aproximar la función ideal desconocida, ya que reduce el número de neuronas si se usa convenientemente. Por lo tanto, una buena heurística sería disponer de un repertorio básico de funciones de activación que contemplen distintas imágenes no polinómicas.

Además de las propuestas en 4.2, añadimos a nuestra colección las siguientes.

⊗ **Función no polinómica**
Que no es un polinomio.

6. Funciones de activación

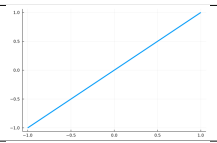
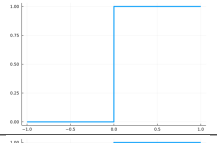
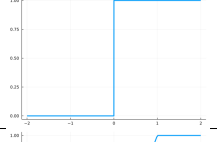
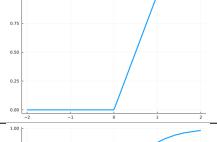
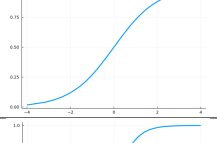
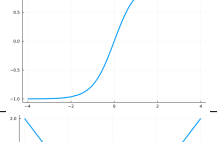
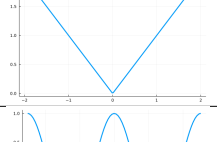
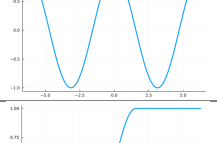
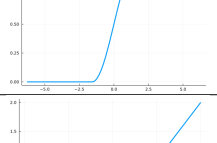
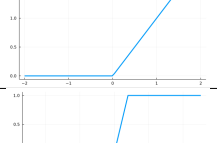
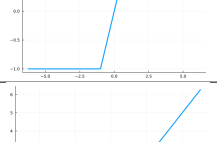
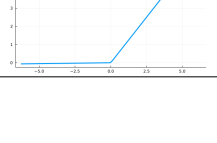
Nombre	Expresión	Rango imagen	Gráfica
Identidad	$Id(x) = x$	$(-\infty, +\infty)$	
Indicadora $\lambda \in \mathbb{R}$	$Indicadora_{\lambda}(x) = 1_{\{x > \lambda\}}$	$\{0, 1\}$	
Función umbral p polinomio	$Threshold(x) = 1_{\{p(x) > 0\}}$	$\{0, 1\}$	
Función rampa	$Rampa(x) = x1_{\{0 < x < 1\}} + 1_{\{x \geq 1\}}$	$[0, 1]$	
Sigmoidea	$\sigma(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$	
Tangente hiperbólica	\tanh	$(-1, 1)$	
Valor absoluto	$abs(x) = x $	$[0, +\infty)$	
Coseno	\cos	$[-1, 1]$	
Cosine Squasher	$CosineSquasher(x) = \left(1 + \cos\left(x + 3\frac{\pi}{2}\right)\frac{1}{2}\right)1_{\{-\frac{\pi}{2} \leq x \leq \frac{\pi}{2}\}} + 1_{\{\frac{\pi}{2} < x\}}$	$[0, 1]$	
ReLU	$ReLU(x) = \max(0, x)$	$[0, +\infty)$	
Hard Hyperbolic Function	$Hardtanh(x) = \begin{cases} -1 & \text{si } x \leq -1 \\ x & \text{si } -1 < x < 1 \\ 1 & \text{si } x \geq 1 \end{cases}$	$[-1, 1]$	
Leaky ReLU	$LReLU_{\alpha}(x) = \begin{cases} \alpha x & \text{si } x \leq 0 \\ x & \text{si } x > 0 \end{cases}$ con $\alpha \in \mathbb{R}^+$ valor pequeño.	$[0, +\infty)$	

Tabla 6.1.: Compendio de funciones de activación

Aportación original

Teorema 6.1. Sea $\phi \in \mathcal{A}(\mathbb{R}^2)$ una transformación afín, sean dos funciones de activación σ, γ tales que

$$\phi \circ \sigma = \gamma,$$

entonces para el espacio de redes neuronales de n neuronas creado con la función de activación σ es igual al espacio de redes neuronales creado con la función de activación γ .

Demostración. Sea $\mathcal{H}_{\sigma,n}^+(\mathbb{R}^d, \mathbb{R}^s)$, el espacio de redes neuronales con n neuronas con sesgo.

Para cualquier $h \in \mathcal{H}_{\sigma,n}^+(\mathbb{R}^d, \mathbb{R}^s)$ la proyección i -ésima de h será de la forma

$$h_i(x) = \sum_{j=1}^n (\beta_j \sigma(A_j(x)) + k_j),$$

con $x \in \mathbb{R}^d, \beta_j, k_j \in \mathbb{R}, A_j \in \mathcal{A}(\mathbb{R}^d)$.

Procedemos a definir $\tilde{h}_i(x)$ como sigue y se tiene que

$$\tilde{h}_i(x) = \sum_{j=1}^n (\beta_j \phi(\sigma(A_j(x))) + k_j) = \sum_{j=1}^n (\tilde{\beta}_j \sigma(\tilde{A}_j(x)) + \tilde{k}_j),$$

con $x \in \mathbb{R}^d, \tilde{\beta}_j, \tilde{k}_j \in \mathbb{R}, \tilde{A}_j \in \mathcal{A}(\mathbb{R}^d)$, por lo que está claro que $\tilde{h}_i(x) \in \mathcal{H}_{\sigma,n}^+(\mathbb{R}^d, \mathbb{R}^s)$.

Pero por hipótesis del teorema $\phi \circ \sigma = \gamma$, por lo que $\tilde{h}_i(x) \in \mathcal{H}_{\gamma,n}^+(\mathbb{R}^d, \mathbb{R}^s)$.

Basta con considerar la transformación afín inversa para ver que ambos espacios son isomorfos

$$\mathcal{H}_{\gamma,n}^+(\mathbb{R}^d, \mathbb{R}^s) \simeq \mathcal{H}_{\sigma,n}^+(\mathbb{R}^d, \mathbb{R}^s).$$

Dos conjuntos isomorfos donde para cualquier $h \in \mathcal{H}_{\gamma,n}^+(\mathbb{R}^d, \mathbb{R}^s)$, se tiene que $h \in \mathcal{H}_{\sigma,n}^+(\mathbb{R}^d, \mathbb{R}^s)$ via el isomorfismo y luego componer con la inversa de la aplicación afín.

Como demostramos en 5.1.1 se tiene que

$$\mathcal{H}_{\sigma,n}^+(\mathbb{R}^d, \mathbb{R}^s) = \mathcal{H}_{\gamma,n}^+(\mathbb{R}^d, \mathbb{R}^s) \subset \mathcal{H}_{\gamma,n+1}^+(\mathbb{R}^d, \mathbb{R}^s) \subset \mathcal{H}_{\sigma,n+1}^+(\mathbb{R}^d, \mathbb{R}^s) = \mathcal{H}_{\sigma,n+1}^+(\mathbb{R}^d, \mathbb{R}^s).$$

Por lo que para un n arbitrariamente grande, se acaba de probar lo buscado.

$$\mathcal{H}_{\gamma}(\mathbb{R}^d, \mathbb{R}^s) = \mathcal{H}_{\sigma}(\mathbb{R}^d, \mathbb{R}^s).$$

□

🔗 Relevancia práctica del teorema

Este teorema lo que nos está diciendo es que si dos funciones de activación tienen *la misma forma* (independientemente de su grafo) entonces **aproximarán igual de bien**, es decir, con el mismo error dentro de un conjunto de datos. Esto a nivel práctico significa que **si se tienen dos funciones de activación con la misma forma (o muy parecidas) elige la que tenga menor costo computacional**, porque a nivel teórico aproximarán igual de bien y de esta manera ahorraremos recursos.

6. Funciones de activación

Notemos además que la demostración nos enseña que la igualdad se da independientemente del número de neuronas fijado, es decir que no es un resultado asintótico (lo asintótico en términos prácticos significa que sea resultado de una serie de aproximaciones).

Así pues a la vista de la imágenes de las distintas funciones de activación recogidas en la tabla 6.1 y por el recién probado teorema 6.1 podemos determinar a priori que de manera teórica existen conjuntos de funciones que aproximadamente puedes producir los mismos resultado, compararemos entonces su coste computacional y tomaremos como representante de la clase aquel que sea de menor coste.

<i>Grupo escalera</i>	<i>Grupo sigmoide</i>	<i>Grupo ReLU</i>
Indicadora Umbral	Rampa Sigmodea <i>Cosine Squasher</i> tanh <i>Hard Hyperbolic Function</i>	ReLU LReLU

Tabla 6.2.: Agrupaciones de funciones de activación con forma similar

6.1.1. Implementación de las funciones de activación en la biblioteca de redes neuronales

Las funciones de activación han sido implementadas con cuidado de que sean eficientes y valiéndose de las características propias de Julia, para ello se han utilizado técnicas como:

- **Programación modular:** Tal y como se recomienda en la documentación de Julia ¹ se ha utilizado una módulo en la implementación de la biblioteca, esto aporta los siguientes beneficios:
 - Los módulos pueden ser precompilados y de esta manera se aceleraría la carga y el tiempo de inicialización.
 - Encapsulamiento de los métodos y facilidades de uso del espacio de nombres, lo cual forma parte de una buena metodología de programación.
- **Macros²:** que permiten sustituciones de código cuando el código es analizado por el compilador, de esta manera, funciones que devuelven otras funciones dependientes de un parámetro se verán beneficiadas. Puede encontrar la implementación de esto en la biblioteca de redes neuronales implementada en nuestro repositorio ³

6.1.2. Coste computacional funciones activación

6.1.2.1. Diseño del experimento

El experimento para comparar los resultados ha consistido: Se ha evaluado cada función a comparar 20000000 veces y se ha medido cuanto tarda. Esto se ha repetido 15 veces. Puede

¹Consultada en la página web oficial de Julia a día 23 de mayo del 2022 con URL: <https://docs.julialang.org/en/v1/manual/modules/>

²La información consultada de macros ha sido de la página oficial de Julia, a día 23 de mayo del 2022, URL: <https://docs.julialang.org/en/v1/manual/metaprogramming>

³Esto es en https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales/tree/main/Biblioteca-Redes-Neuronales/src/activation_functions.jl

encontrar la implementación concreta y los resultados concretos de cada iteración en el repositorio del proyecto ⁴.

6.1.2.2. Test de hipótesis

Compararemos si los resultados son significativos utilizando la **prueba de los rangos con signo de Wilcoxon** (véase [DD22], [McD14], o la web de [cienciadedatos.net](https://www.cienciadedatos.net) ⁵).

La motivación de realizar esta prueba es la siguiente:

- Las muestras son independientes.
- Los datos tomados, el tiempo, permiten ser ordenados.
- El tamaño de muestra es pequeño y no podemos asegurar normalidad de la datos.

Hipótesis

- H_0 : La mediana de las diferencia de cada par de datos es 0.
- H_a : La mediana de las diferencia entre cada par de datos es diferente de cero.

La utilidad de este test es que si rechaza la hipótesis la hipótesis nula sabremos que con un 95 % de certeza tendrán medianas diferentes, es decir, **existe una diferencia de tiempos**. En caso de que no se rechace no podremos afirmar nada. Puede encontrar la implementación en el repositorio del proyecto ⁶.

Los resultados del test de Wilcoxon han sido los siguientes:

	cte 1	Identidad	Threshold de 2x	CosineSquasher	Indicadora de o
cte 1	-	No rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Identidad	No rechaza H_0	-	Rechaza H_0	Rechaza H_0	Rechaza H_0
Threshold de 2x	Rechaza H_0	Rechaza H_0	-	Rechaza H_0	No rechaza H_0
CosineSquasher	Rechaza H_0	Rechaza H_0	Rechaza H_0	-	Rechaza H_0
Indicadora de o	Rechaza H_0	Rechaza H_0	No rechaza H_0	Rechaza H_0	-
Rampa	Rechaza H_0	Rechaza H_0	No rechaza H_0	Rechaza H_0	Rechaza H_0
ReLU	Rechaza H_0	Rechaza H_0	No rechaza H_0	Rechaza H_0	No rechaza H_0
Sigmoid	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Tangente hiperbólica	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Valor absoluto	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Coseno	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Hardtanh	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
LReLU	Rechaza H_0	Rechaza H_0	No rechaza H_0	Rechaza H_0	No rechaza H_0

Tabla 6.3.: Resultados 1 de 3: Rechazos con un 95 % de confianza en el test Wilcoxon.

⁴En el directorio de experimentos de <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales>.

⁵Prueba de los rangos con signo de Wilcoxon by Joaquín Amat Rodrigo, available under a Attribution 4.0 International (CC BY 4.0) at https://www.cienciadedatos.net/documentos/18_prueba_de_los_rangos_con_signo_de_wilcoxon Con fecha de visita el 22 de mayo del 2022.

⁶En el directorio de experimentos de <https://github.com/BlancaCC/TFG-Estudio-de-las-redes-neuronales>.

6. Funciones de activación

	Rampa	ReLU	Sigmoid	Tangente hiperbólica
cte 1	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Identidad	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Threshold de $2x$	No rechaza H_0	No rechaza H_0	Rechaza H_0	Rechaza H_0
CosineSquasher	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Indicadora de 0	Rechaza H_0	No rechaza H_0	Rechaza H_0	Rechaza H_0
Rampa	-	No rechaza H_0	Rechaza H_0	Rechaza H_0
ReLU	No rechaza H_0	-	Rechaza H_0	Rechaza H_0
Sigmoid	Rechaza H_0	Rechaza H_0	-	Rechaza H_0
Tangente hiperbólica	Rechaza H_0	Rechaza H_0	Rechaza H_0	-
Valor absoluto	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Coseno	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Hardtanh	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
LReLU	No rechaza H_0	No rechaza H_0	Rechaza H_0	Rechaza H_0

Tabla 6.4.: Resultados 2 de 3: Rechazos con un 95 % de confianza en el test Wilcoxon.

	Valor absoluto	Coseno	Hardtanh	LReLU
cte 1	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Identidad	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Threshold de $2x$	Rechaza H_0	Rechaza H_0	Rechaza H_0	No rechaza H_0
CosineSquasher	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Indicadora de 0	Rechaza H_0	Rechaza H_0	Rechaza H_0	No rechaza H_0
Rampa	Rechaza H_0	Rechaza H_0	Rechaza H_0	No rechaza H_0
ReLU	Rechaza H_0	Rechaza H_0	Rechaza H_0	No rechaza H_0
Sigmoid	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Tangente hiperbólica	Rechaza H_0	Rechaza H_0	Rechaza H_0	Rechaza H_0
Valor absoluto	-	Rechaza H_0	Rechaza H_0	No rechaza H_0
Coseno	Rechaza H_0	-	Rechaza H_0	Rechaza H_0
Hardtanh	Rechaza H_0	Rechaza H_0	-	Rechaza H_0
LReLU	Rechaza H_0	No rechaza H_0	Rechaza H_0	-

Tabla 6.5.: Resultados 3 de 3: Rechazos con un 95 % de confianza en el test Wilcoxon.

Como ya comentábamos, si la hipótesis nula es rechazada podemos suponer que hay una diferencia de tiempo significativas; en caso contrario no podemos saber nada.

Sin embargo podemos entender estos rechazos como una clase de equivalencia; es decir, la diferencia en coste computacional no es tan significativa, dentro de ese grupo. De hecho, como podemos apreciar en la tabla 6.6, que está ordenada de menor tiempo a mayor, estos se encuentra en posiciones consecutivas.

6.1. Caracterización de las funciones de activación

Función	Mediana	Media Tiempo	Desviación típica
cte 1 (para comparar)	1475,959	1473,478	26,332
Identidad (para comparar)	1479,817	1467,311	27,021
Hardtanh	1495,105	1491,046	21,334
CosineSquasher	1522,128	1521,117	19,223
ReLU	1546,379	1552,049	21,435
Indicadora de 0	1554,432	1556,114	21,814
Rampa	1557,449	1552,169	25,043
Threshold de $2x$	1562,809	1556,669	23,029
LReLU	1564,124	1561,367	21,722
Valor absoluto	1583,266	1580,545	23,464
Sigmoid	1608,797	1601,079	21,938
Coseno	1630,392	1629,634	26,113
Tangente hiperbólica	1664,006	1653,295	23,025

Tabla 6.6.: Tiempo de ejecución en segundos

Si volvemos a nuestro objetivo, que era encontrar el representante de menor costo entre las agrupaciones dispuestas en la tabla 6.2. Concluimos que los mejores candidatos son:

- Para el *grupo escalera*: No se ha rechazado la hipótesis nula, luego a priori no hay deferencia significativa y podemos seleccionar el candidato que queramos.
- Para el *grupo sigmoide*: La mejor opción ha sido *Hard Hyperbolic Function* y después en orden de mejor a peor: *Cosine Squasher*, *rampa*, *sigmodea* y *tangente hiperbólica*.
- Pare el *grupo ReLU*: No se ha rechazado la hipótesis nula, así que no podemos decir nada.

7. Hipótesis de optimización

ATENCIÓN: Todos este capítulo está como notas personales

ATENCIÓN: Todo este capítulo está como notas personales

En esta sección recopilaremos las posibles ideas que podrían optimizar las redes neuronales, describiremos una experimentación para contrastar los resultados y mostraremos sus conclusiones.

Vamos a tratar de encontrar de encontrar un ejemplo de esto.

7.1. Inicialización de la pesos red neuronal

7.2. Algoritmo genéticos

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

- [43087] Ieee first annual international conference on neural networks san diego, california june 21-24, 1987. *IEEE Expert*, 2(2):14–14, 1987. doi:10.1109/MEX.1987.4307059. [Citado en pág. 41]
- [ADIP20] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, and Roberto Prevete. A survey on modern trainable activation functions. *CoRR*, abs/2005.00817, 2020. URL: <https://arxiv.org/abs/2005.00817>, arXiv:2005.00817. [Citado en pág. 95]
- [AMMIL12] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012. [Citado en págs. 7, 24, 25, 40, 77, and 85]
- [Bar47] Robert G. Bartle. *The elements of real analysis*. John. Wiley & Sons, 1947. [Citado en pág. 27]
- [BCC] Francisco Javier Merí de la Maza Blanca Cano Camarero, Juan Julián Merelo Guervós. Github, repositorio: Estudio de las redes neuronales. URL: <https://github.com/BlancaC/TFG-Estudio-de-las-redes-neuronales>. [Citado en pág. 17]
- [BGNR16] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016. URL: <http://arxiv.org/abs/1611.02167>, arXiv:1611.02167. [Citado en pág. 25]
- [Biso6] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. [Citado en págs. 25, 40, and 71]
- [Bos] Geert Bossuyt. Agile is not a methodology, it's a mindset ! URL: <https://xebia.com/agile-is-not-a-methodology-its-a-mindset/>. [Citado en pág. 14]
- [BPS⁺22] Atılım Güneş Baydin, Barak A. Pearlmutter, Don Syme, Frank Wood, and Philip Torr. Gradients without backpropagation. 2022. URL: <https://arxiv.org/abs/2202.08587>, doi:10.48550/ARXIV.2202.08587. [Citado en pág. 87]
- [BS21] Sebastien Bubeck and Mark Sellke. A universal law of robustness via isoperimetry. In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL: <https://openreview.net/forum?id=z710SKqTFh7>. [Citado en pág. 23]
- [CC95] Tianping Chen and Hong Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995. doi:10.1109/72.392253. [Citado en pág. 43]
- [CM07] Vladimir Cherkassky and Filip Mulier. *Learning From Data: Concepts, Theory, and Methods*. John. Wiley & Sons, 2 edition, 2007. [Citado en págs. 24, 76, and 80]
- [Coh09] Mike Cohn. *User Stories for Agile Software*. Pearson Education, Inc., 2009. [Citado en pág. 14]
- [CZLN21] Junyi Chai, Hao Zeng, Anming Li, and Eric W.T. Ngai. Deep learning in computer vision: A critical review of emerging techniques and application scenarios. *Machine Learning with Applications*, 6:100134, 2021. URL: <https://www.sciencedirect.com/science/article/pii/S2666827021000670>, doi:https://doi.org/10.1016/j.mlwa.2021.100134. [Citado en pág. 23]

Bibliografía

- [D] Rikke Friis Dam and Teo Yu Siang |. Personas – a simple introduction. URL: <https://www.interaction-design.org/literature/article/personas-why-and-how-you-should-use-them>. [Citado en pág. 13]
- [DD22] Christopher D Barr David Diez, Mine Çentinkaya-Runder. *OpenIntro Statics*. Openintro, 2022. URL: <https://github.com/OpenIntroStat/openintro-statistics>. [Citado en pág. 99]
- [Dug66] James Dugundji. *Topology*. Allyn and Bacon series in advanced mathematics. Allyn and Bacon, 1966. URL: <https://books.google.es/books?id=FgFRAAAAMAAJ>. [Citado en pág. 58]
- [DZDW18] Happiness Ugochi Dike, Yimin Zhou, Kranthi Kumar Deveerasetty, and Qingtian Wu. Unsupervised learning based on artificial neural network: A review. In *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*, pages 322–327, 2018. doi: [10.1109/CBS.2018.8612259](https://doi.org/10.1109/CBS.2018.8612259). [Citado en pág. 25]
- [Fun89] Ken-Ichi Funahashi. On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192, 1989. URL: <https://www.sciencedirect.com/science/article/pii/0893608089900038>, doi:[https://doi.org/10.1016/0893-6080\(89\)90003-8](https://doi.org/10.1016/0893-6080(89)90003-8). [Citado en págs. 43 and 95]
- [GCZ⁺19] Shanghua Gao, Ming-Ming Cheng, Kai Zhao, Xinyu Zhang, Ming-Hsuan Yang, and Philip H. S. Torr. Res2net: A new multi-scale backbone architecture. *CoRR*, abs/1904.01169, 2019. URL: <http://arxiv.org/abs/1904.01169>, arXiv:1904.01169. [Citado en pág. 88]
- [Gue] Juan Julián Merelo Guervós. Cómo llevar a término un tfg/tfm en informática. URL: <https://jj.github.io/que-es-un-trabajo-fin-de-x/xf.html>. [Citado en pág. 13]
- [Gue21] Juan Julián Merelo Guervós. Agile (data) science: a (draft) manifesto. *CoRR*, abs/2104.12545, 2021. URL: <https://arxiv.org/abs/2104.12545>, arXiv:2104.12545. [Citado en pág. 13]
- [GW88] A. Ronald Gallant and Halbert White. There exists a neural network that does not make avoidable mistakes, 1988. [Citado en pág. 43]
- [Hal74] Paul R. Halmos. *Measure theory / [by] Paul R. Halmos*. Springer-Verlag New York, 1974. URL: <http://www.loc.gov/catdir/enhancements/fy0814/74010690-t.html>. [Citado en págs. 43, 49, 58, and 60]
- [His21] Hisour. Nocción del aprendizaje automático, 2021. URL: <https://www.hisour.com/es/machine-learning-42773/>. [Citado en pág. 24]
- [HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. URL: <https://www.sciencedirect.com/science/article/pii/0893608089900208>, doi:[https://doi.org/10.1016/0893-6080\(89\)90020-8](https://doi.org/10.1016/0893-6080(89)90020-8). [Citado en págs. 23, 41, and 43]
- [Ken] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas. Principios del manifiesto Ágil. URL: <https://agilemanifesto.org/iso/es/principles.html>. [Citado en pág. 14]
- [LCL⁺21] Tingting Liang, Xiaojie Chu, Yudong Liu, Yongtao Wang, Zhi Tang, Wei Chu, Jingdong Chen, and Haibin Ling. Cbnetv2: A composite backbone network architecture for object detection. *CoRR*, abs/2107.00420, 2021. URL: <https://arxiv.org/abs/2107.00420>, arXiv:2107.00420. [Citado en pág. 88]
- [LGMC21] Luis Liñán-Villafranca, Mario García-Valdez, Juan Julián Merelo, and Pedro Castillo-Valdivieso. Evomlp: A framework for evolving multilayer perceptrons. In Ignacio Rojas, Gonzalo Joya, and Andreu Català, editors, *Advances in Computational Intelligence - 16th International Work-Conference on Artificial Neural Networks, IWANN 2021, Virtual Event, June*

- 16-18, 2021, *Proceedings, Part II*, volume 12862 of *Lecture Notes in Computer Science*, pages 330–342. Springer, 2021. doi:10.1007/978-3-030-85099-9_27. [Citado en pág. 23]
- [Lig58] M. J. Lighthill. *Introduction to Fourier Analysis and Generalized Functions*. Cambridge University Press, 1958. [Citado en pág. 87]
- [LKC⁺10] Victor Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupati, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. Debunking the 100x gpu vs. cpu myth: an evaluation of throughput computing on cpu and gpu. *ACM SIGARCH Computer Architecture News*, 38:451–460, 01 2010. doi:10.1145/1816038.1816021. [Citado en pág. 69]
- [McD14] J.H. McDonald. *Handbook of Biological Statistics*. Sparky House Publishing, Baltimore, Maryland, 2014. [Citado en pág. 99]
- [Mit97] Tom Mitchell. *Machine Learning*. McGraw, 1997. URL: <https://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/mlbook.html>. [Citado en pág. 24]
- [Mono2] David J. Montana. Neural network weight selection using genetic algorithms. 2002. [Citado en pág. 88]
- [NIGM18] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *CoRR*, abs/1811.03378, 2018. URL: <http://arxiv.org/abs/1811.03378>, arXiv:1811.03378. [Citado en pág. 46]
- [PKK⁺19] Michael P. Perrone, Haidar Khan, Changhoan Kim, Anastasios Kyrillidis, Jerry Quinn, and Valentina Salapura. Optimal mini-batch size selection for fast gradient descent. *CoRR*, abs/1911.06459, 2019. URL: <http://arxiv.org/abs/1911.06459>, arXiv:1911.06459. [Citado en pág. 87]
- [RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. doi:10.1038/323533a0. [Citado en pág. 80]
- [Rud] Charlie Rudd. Why agile is not a methodology. URL: <https://www.solutionsiq.com/resource/blog-post/why-agile-is-not-a-methodology/>. [Citado en pág. 14]
- [Sar21] Iqbal H. Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN Computer Science*, 2(3):160, 2021. doi:10.1007/s42979-021-00592-x. [Citado en págs. 23 and 25]
- [SM02] Kenneth O. Stanley and Risto Miikkulainen. Efficient reinforcement learning through evolving neural network topologies. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, GECCO'02*, page 569–577, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. [Citado en pág. 25]
- [SM15] Sho Sonoda and Noboru Murata. Neural network with unbounded activations is universal approximator. *CoRR*, abs/1505.03654, 2015. URL: <http://arxiv.org/abs/1505.03654>, arXiv:1505.03654. [Citado en págs. 43 and 95]
- [SVI⁺15] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. URL: <http://arxiv.org/abs/1512.00567>, arXiv:1512.00567. [Citado en pág. 46]
- [WLH⁺17] Bin Wang, Tianrui Li, Yanyong Huang, Huaishao Luo, Dongming Guo, and Shi-Jinn Horng. Diverse activation functions in deep learning. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*, pages 1–6, 2017. doi:10.1109/ISKE.2017.8258768. [Citado en pág. 46]
- [WY10] Bogdan M. Wilamowski and Hao Yu. Neural network learning without backpropagation. *IEEE Transactions on Neural Networks*, 21(11):1793–1803, 2010. doi:10.1109/TNN.2010.2073482. [Citado en pág. 88]

Agradecimientos

Agradezco a