Escuela Técnica Superior de Ingenieros Industriales
Universidad Politécnica de Madrid

José Gutiérrez Abascal, 2. 28006 Madrid
Tel.: 91 336 3060
info.industriales@upm.es

www.industriales.upm.es

Víctor Pastor Ruiz

05 TRABAJO FIN DE GRADO

INDUSTRIALES

TRABAJO FIN DE GRADO

DESARROLLO DE REDES NEURONALES PARA RESOLUCIÓN DE PROBLEMAS DE ESTRUCTURAS

TRABAJO FIN DE GRADO PARA LA OBTENCIÓN DEL TÍTULO DE **GRADUADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES**

SEPTIEMBRE **2021**

Víctor Pastor Ruiz

DIRECTOR DEL TRABAJO FIN DE GRADO:
David Portillo García

Escuela Técnica Superior de Ingenieros Industriales
Universidad Politécnica de Madrid
Madrid, Septiembre de 2021

DESARROLLO DE REDES NEURONALES PARA RESOLUCIÓN DE PROBLEMAS DE ESTRUCTURAS

Trabajo Fin de Grado
Grado en Ingeniería en Tecnologías Industriales

Autor: Víctor Pastor Ruiz
Director: David Portillo García

Agradecer al tutor, David Portillo, por ofrecerme un tema atrevido e interesante, que me suscitaba gran interés, y por su ayuda y dedicación al proyecto. Agradecer a Kaustav Sarkar y Manu, por sacarme de un pozo sin resultados. Por último agradecer a mi familia, amigos y a mi pareja, por su apoyo incondicional.

Resumen

La inteligencia artificial es una de las tendencias tecnológicas más candentes de la actualidad. En particular, se están encontrando numerosas aplicaciones donde las redes neuronales suponen una solución particularmente interesante. Los casos más relevantes son: el reconocimiento de imágenes, la conducción autónoma o la generación de texto escrito. A pesar del misticismo que las rodea, su funcionamiento no depende de trucos de magia, más bien de herramientas matemáticas conocidas por cualquier estudiante en un curso básico de cálculo: funciones lineales, derivadas parciales, regla de la cadena, gradiente ...

Las redes neuronales son especialmente interesantes por poder adaptarse a cualquier conjunto de datos, por muy complejo que sea, no obstante, existen soluciones más convencionales con las mismas características, por ejemplo, los modelos polinómicos. La independencia de la morfología de los datos de entrada, la velocidad de aprendizaje o la mejor capacidad de generalizar, son algunas de las ventajas que poseen las redes neuronales sobre estos modelos clásicos.

Si se enfrenta a una red neuronal contra un problema de vigas, esta será capaz de aprender las leyes físicas subyacentes que lo definen, no obstante, el algoritmo de minimización que se use durante el aprendizaje marcará la diferencia entre el éxito y el fracaso, comprobándose que el descenso del gradiente no supone una solución adecuada para este tipo de problemas. En este caso se encuentra que el algoritmo de minimización de Levenberg-Marquardt ofrece unos resultados mucho mejores. Dicho algoritmo supone una mezcla entre el descenso del gradiente ya comentado y el ampliamente conocido método de Newton. No obstante, la selección de la arquitectura y ajuste de hiperparámetros es un mundo muy extenso y algo desconocido, por lo que encontrar soluciones muy óptimas requiere de un proceso de prueba y error, que consume mucho tiempo y recursos.

Para el entrenamiento de estos modelos será imprescindible contar con grandes conjuntos de datos con una gran número de parejas inputs/outputs a las que se amolden. La elaboración manual de estos conjuntos es una labor larga y tediosa, siendo la herramienta smmatlab la solución perfecta para generar los datasets. Desarrollado con Matlab, y programado con clases, podrá generar las soluciones de cualquier problema con vigas. Las clases representarán los distintos elementos que componen la estructura, por ejemplo se cuenta con la clase viga, que contiene la información referente al material que la compone, su sección, la curva directora o las coordenadas de principio y fin. Otros ejemplos de clases describen los apoyos o las conexiones. La construcción de las ecuaciones y su resolución se hará mediante el uso del

cálculo simbólico de Matlab, además se podrá elegir entre dos tipos de resoluciones, usando las ecuaciones clásicas de equilibrio y compatibilidad o mediante minimización de la energía elástica de la estructura.

Para la resolución de los problemas con las redes, se debe definir previamente un marco de referencia sobre el significado de los datos de entrada, de modo que en el vector de entrada a la red, cada posición otorgue siempre la misma información y la red pueda aprender. Cuando se trata de problemas específicos, es fácil establecer los grados de libertad del problema y definir la forma de los vectores de entrada. Sin embargo, en caso de que se quiera entrenar una red neuronal mucho más amplia y potente, capaz de resolver muchos tipos de problemas, se debe definir una estructura de datos de entrada más genérica. Una forma de hacerlo sería discretizar la estructura en un número máximo de puntos, y por cada punto fijar un número de variables máximas necesarias para definir todo lo que ocurre en dicho punto, su posición, existencia de viga, apoyo o conexión, material, sección ... De esta manera, con los ordenadores adecuados, se podría entrenar una red con cientos de miles de entradas para resolver cualquier tipo de estructura.

Palabras clave
Inteligencia Artificial, redes neuronales, descenso del gradiente, Levenberg-Marquardt, Matlab, vigas, backpropagation.

Índice general

1. Introducción **1**
1.1. Estado de la literatura . . . 1
1.2. Desde la inteligencia artificial al machine learning . . . 2
1.2.1. Aprendizaje supervisado . . . 2
1.2.2. Aprendizaje reforzado . . . 3
1.3. La mejora del hardware . . . 3
1.4. Objetivos . . . 5

2. Redes neuronales artificiales **7**
2.1. La neurona . . . 7
2.2. Funciones de activación . . . 7
2.2.1. Unidad lineal rectificada . . . 8
2.2.2. Tangente hiperbólica . . . 8
2.2.3. Sigmoide . . . 9
2.3. La red . . . 9
2.4. Backpropagation . . . 10
2.5. Descenso del gradiente . . . 12
2.6. Modificaciones al descenso del gradiente clásico . . . 13
2.6.1. Stochastic Gradient Descent with Momentum (SGDM) . . . 13
2.6.2. RMSProp . . . 14
2.6.3. Adam . . . 14
2.7. Levenberg-Marquardt . . . 14
2.8. Regularización . . . 15
2.9. Teorema de aproximación universal . . . 16
2.9.1. Deep learning . . . 17

3. smMatlab y generación de datasets **18**
3.1. smMatlab . . . 18
3.1.1. Clase `support` . . . 19
3.1.2. Clase `connection` . . . 20
3.1.3. Clase `beam` . . . 20
3.1.4. Clase `structure` . . . 21
3.1.5. Clase `smProblem` . . . 22
3.1.6. Ejemplos . . . 22
3.2. Viga con esfuerzos constantes . . . 24

3.3. Viga simplemente apoyada . . . 26

4. Aproximación polinómica **28**
4.1. Aproximación de funciones . . . 28
4.1.1. Aproximación polinómica . . . 30
4.1.2. Teorema de Stone-Weirstrass . . . 31
4.2. Viga con esfuerzos constantes . . . 32
4.2.1. Sin modificar . . . 33
4.2.2. Modificado . . . 36
4.3. Viga simplemente apoyada . . . 39

5. Descenso del gradiente vs Levenberg-Marquardt **42**
5.1. Viga con esfuerzos constantes . . . 42
5.1.1. Descenso del gradiente . . . 42
5.1.2. Levenberg-Marquardt . . . 46
5.2. Viga simplemente apoyada . . . 49
5.2.1. Descenso del gradiente . . . 49
5.2.2. Levenberg-Marquardt . . . 52
5.3. Resolución de ejemplos . . . 54

6. Resolución de un problema de varias cargas con una red neuronal **56**

7. Diseño de una red neuronal para una estructura genérica **60**
7.1. Introducción . . . 60
7.2. Estructura fija . . . 60
7.3. Estructura genérica . . . 61
7.3.1. Salida de datos . . . 64

8. Conclusiones y lineas futuras **65**
8.1. Conclusiones . . . 65
8.2. Lineas futuras . . . 66

A. Código fuente **67**

B. Planificación y presupuesto **76**

Lista de figuras **81**

Lista de tablas **83**

Capítulo 1

Introducción

El presente trabajo de fin de grado tratará de embarcar al lector en un viaje por el campo del **machine learning**, desde un conocimiento general de las distintas técnicas y algoritmos, pasando por conocer las entrañas de las redes neuronales y los pilares de su funcionamiento hasta su aplicación práctica en un problema de Ingeniería Mecánica, el cálculo de estructuras.

1.1. Estado de la literatura

En el campo de la inteligencia artificial y el machine learning la literatura es extensa. Desde los modelos de neurona de McCulloch y Pitts de la mitad del siglo pasado, hasta obras más modernas como el Murphy [1]. Los modelos aquí elaborados están muy lejos de la complejidad de los modelos en estado del arte, no obstante, se intentan aprovechar los últimos hallazgos en la materia.

En el área de investigación, se han desarrollado ya desde hace tiempo técnicas para extraer las leyes físicas a partir de datos [2]. Especial importancia han tomado los sistemas conservativos, capaces de modelar una gran cantidad de fenómenos y cuya modelización hamiltoniana permite obtener métodos más rápidos que permitan la interacción en tiempo real [3],[4], [5]. En particular, en el área de la Ingeniería mecánica, existe una gran cantidad de trabajos en el área de optimización de control. Estos trabajos se enmarcan dentro del desarrollo de algoritmos capaces de predecir el comportamiento de ciertas máquinas, habiendo cobrado especial interés en los últimos años el control autónomo de drones.

Más concretamente en el área de estructuras y mecánica de sólidos, el desarrollo es más moderno y recientemente se ha introducido el llamado *data driven computing* para el modelado de materiales a partir de datos, dejando a un lado la parte de modelización mediante una ley constitutiva del material, y que juntándolo con métodos numéricos más tradicionales, como elementos finitos o diferencias finitas, dan lugar a métodos robustos de simulación [6], [7]. Recientemente, en [8] se desarrolla un método basado en redes neuronales para encontrar modelos termodinámicos disipativos de no equilibrio (lo que incluye, por ejemplo modelos de materiales viscoelásticos) a partir de datos y conservando las dos primeras leyes de la termodinámica.

1.2. Desde la inteligencia artificial al machine learning

Desde mediados del siglo XX con la evolución de la electrónica y la aparición de los primeros circuitos integrados, surge un nuevo campo tecnológico, un campo cuyo objetivo será el de crear máquinas que imiten los comportamientos inteligentes propios del ser humano, **la Inteligencia Artificial** o IA [9]. Desde una sencilla suma en una calculadora convencional hasta un complejo robot de control en una gran fábrica, esta disciplina ha evolucionado y se ha refinado hasta estar presenten en la mayoría de máquinas y dispositivos de nuestro alrededor.

Hasta hace algún tiempo, la mayoría de comportamientos inteligentes más complejos estaban supeditados a que el ser humano conociese de antemano el problema, la solución y el método para llegar hasta ella, además de saber programar dicho método. Llegados a este punto puede parecer absurdo programar máquinas para realizar una tarea que ya sabemos hacer, no obstante, la velocidad de procesamiento y la gran capacidad para realizar operaciones iterativas ha permitido su aplicación a tareas de gran utilidad. Dicho esto, se podría seguir considerando que este tipo máquinas se quedan algo cortas en su objetivo de imitar el comportamiento inteligente en toda su extensión, es por eso que se debe complementar con otro tipo de técnicas.

El aprendizaje automático o machine learning, ML a partir de ahora, surgió no mucho después que los primeros algoritmos de IA, sin embargo, no ha sido hasta la última década que se ha conseguido obtener todo su potencial, gracias a la mejora del hardware. Los algoritmos de ML tienen como particularidad que son capaces de mejorarse a sí mismos en su comportamiento, es decir, **aprender**. El aprendizaje es de vital importancia en muchos escenarios. Por ejemplo, puede ser que el problema en cuestión sea fácil de resolver, sin embargo, extremadamente difícil de programar, como el reconocimiento de imágenes o la conducción autónoma. Otro ejemplo es que directamente no se conozca como se resuelve. El primero de los escenarios está asociado con el aprendizaje supervisado, el segundo con el aprendizaje no supervisado y reforzado.

1.2.1. Aprendizaje supervisado

El aprendizaje supervisado es el más común entre los tipos de ML. El objetivo del algoritmo es aprender a relacionar un conjunto de parejas de entrada y salida, lo que se conoce como conjunto de entrenamiento. El conjunto de entrenamiento se define como $D = \{(x_i, y_i)\}_{i=1}^{N}$ [1]. x_i es un vector de cualquier dimensión, y representa los atributos o variables de cada observación, puede representar píxeles en imágenes, variables físicas o letras entre otras cosas. y_i es la salida esperada por cada vector x_i, al igual que la entrada, puede ser multidimensional. Las variables de salida pueden ser continuas, si se trata de un problema de regresión, o discretas si se está realizando una clasificación.

1.2.2. Aprendizaje reforzado

En el aprendizaje no supervisado se cuenta con un conjunto de valores sin relación entrada-salida, y el algoritmo deberá de descubrir relaciones entre las variables. Sin embargo, su alcance y repercusión es limitado, por lo que no se discutirá en mayor detalle.

En cambio, existe un aproximación intermedia, denominada aprendizaje reforzado [10]. En el entrenamiento de los algoritmos de aprendizaje reforzado no se cuenta con una salida óptima para cada entrada, pero si que se puede contabilizar si las salidas que está dando el algoritmo están consiguiendo el objetivo y se están acercando a la solución óptima, lo que se conoce como función de recompensa. Un ejemplo práctico es la inteligencia artificial AlphaGo [11]. El Go es un antiguo juego de mesa chino, la particularidad es que en cada turno el jugador tendrá que decidirse entre unas 200 posibilidades de movimientos posibles, siendo muy complicado determinar analíticamente cual es el mejor de todos. Con un algoritmo de aprendizaje reforzado se podría entrenar a un ordenador para que a base de jugar un gran número de partidas, aprendiese a jugar el mejor movimiento posible en cada situación. Usando este método AlphaGo consiguió derrotar a Lee Sedol, uno de los mejores jugadores del mundo de Go.

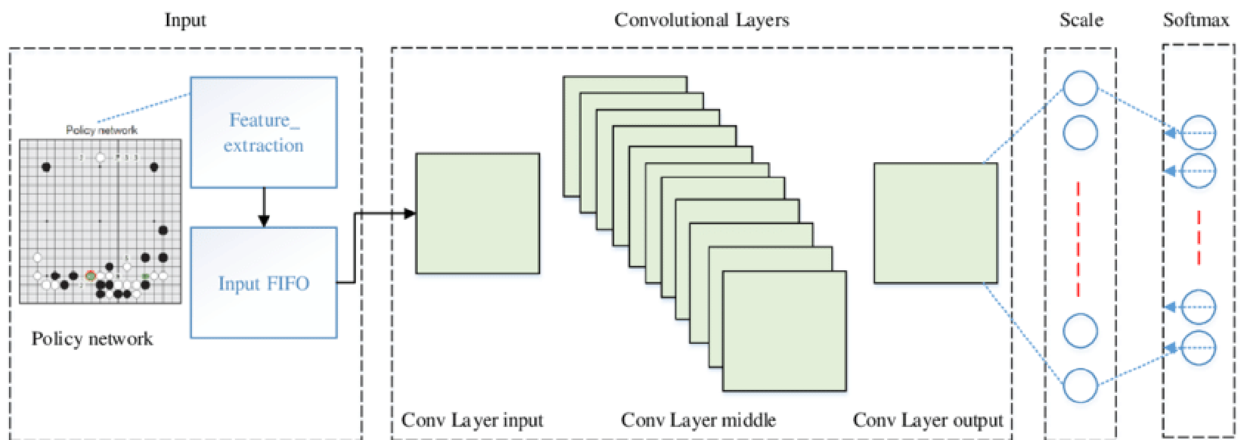


Figura 1.2.1: Arquitectura de la red neuronal de AlphaGo, toma como entrada una matriz que simboliza el tablero y consiste en 13 capas convolucionales, 12 funciones ReLu y una capa softmax. Ref: Li, Zhen-Ni —& Zhu, Can & Yu-Liang, Gao & Wang, Ze-Kun & Wang, Jiao. (2020). AlphaGo Policy Network: A DCNN Accelerator on FPGA. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.3023739.

1.3. La mejora del hardware

En el punto anterior se dice que el aspecto clave que ha dejado prosperar a esta tecnología está relacionado con la mejora del hardware. Las redes neuronales se llevan desarrollando desde hace décadas, no obstante, solo estaban respaldadas por un potencial teórico, pero que no daba resultados en la práctica. Resulta que nos estábamos quedando cortos con la

cantidad de datos que se necesitaban para entrenarlas. A día de hoy a todo el mundo le suena el 'Big Data', y es que en esta última década ha sido uno de los términos de moda, y no por casualidad, ya que actualmente en internet circula una cantidad masiva de datos en todo momento, y es que ahora poca gente se imagina vivir sin un ordenador o sin un teléfono móvil, el internet se ha convertido en un objeto esencial de nuestras vidas y con ello los datos que en él circulan. En el gráfico de la figura 1.3.1 se aprecia como se fueron incrementando las búsquedas en Google hasta el año 2012, llegando a 1.2 trillones de búsquedas dicho año, si bien este número es muy grande, se queda pequeño sabiendo que en los primeros 8 meses de 2021 ya llevamos más de 2 trillones de búsquedas, llegando a las 4500 millones en las ultimas 13 horas [12]. Pero no solo han incrementado las búsquedas en Google, también el número de fotos subidas a Facebook o de artículos escritos en internet, todos estos datos han supuesto un elemento clave para el entrenamiento de modelos complejos. Las primeras redes no hacían más que diferenciar fotos de gatos y de perros [13], pero con el tiempo, se ha aumentado la complejidad, funcionalidades como la de reconocer peatones en vídeos en movimiento o generar código de programación a partir de la descripción textual del mismo [14] son ahora posibles. La inundación de datos en la red accesibles por todos ha venido acompañada de otros factores clave, como la computación en la nube bajo demanda, que ha permitido acceder a ordenadores masivos de forma remota o la inversión de grandes empresas en crear equipos de personas dedicadas a estos modelos, como es el caso de Google DeepMind, desarrolladores de AlphaGo.

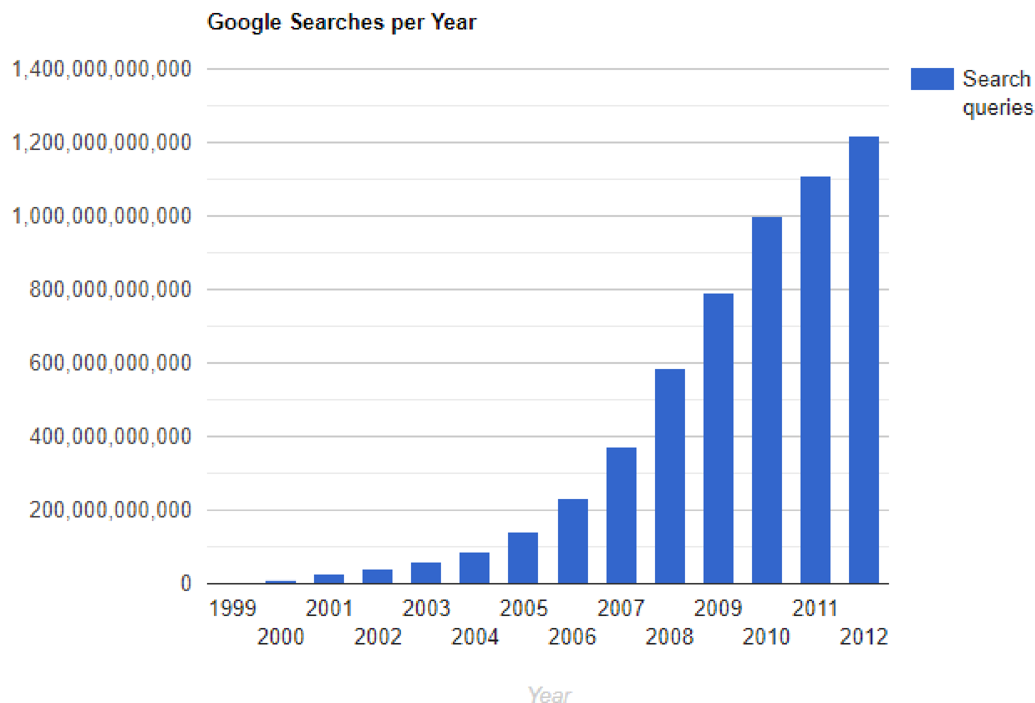


Figura 1.3.1: Búsquedas al año en Google. Ref: Google Search Statistics - Internet Live Stats. (s. f.). Internet Live Stats. Recuperado 4 de septiembre de 2021, de https://www.internetlivestats.com/google-search-statistics/

1.4. Objetivos

Para finalizar con la introducción se va a describir capítulo a capítulo los contenidos del trabajo y se va a realizar una lista de los objetivos que se pretenden durante la realización del documento. En el primer capítulo tras la introducción, se realizará una introducción a las redes neuronales, para luego describir cada unas de sus partes y su funcionalidad, explicando los pormenores matemáticos que las constituyen y algunas de las técnicas y algoritmos que posibilitan su funcionamiento, además, en la última parte, se pretende mostrar el gran potencial de las redes neuronales a través del teorema de aproximación universal. En los siguientes capítulos se abordarán los temas prácticos del trabajo.

El capítulo 3 supondrá una introducción al funcionamiento del programa Smmatlab, herramienta básica para generar los datos necesarios, para posteriormente describir los problemas que se van a tratar de resolver y la morfología de los datos que se van a usar. Los problemas que se van a tratar se introducen a continuación:

- Viga empotrada en un extremo sometida a un momento flector y torsor y a una carga en dirección axial en el extremo libre.
- Viga simplemente apoyada con una carga en un punto intermedio en dirección axial y transversal.

El capítulo 4 comienza con una vuelta a los detalles teóricos, para introducir la técnica de aproximación polinómica, que se va a usar como propuesta alternativa para la resolución de los problemas, y a su vez para compararla con las técnicas de Machine Learning, que son el enfoque principal del trabajo. En este capítulo se expondrán también los resultados al tratar de resolver por esta técnica los problemas propuestos.

A partir de este punto se vuelve a centrar toda la atención en las redes neuronales. En el capítulo 5 se resolverán los problemas con redes neuronales, usando dos algoritmos de minimización distintos, explicados en la teoría: El descenso del gradiente con momento y el método de Levenberg-Marquardt. Además se compararán los resultados que otorgan en un ejemplo concreto, también con los resultados que daría la resolución analítica por Euler–Bernoulli. En el siguiente capítulo se entrenará otra red neuronal, esta vez el problema será el de una estructura concreta, sometida a varias cargas, que cambiarán de magnitud y posición. Se introduce un problema de esta características, con el objetivo de darle un enfoque más cercano a algo que se podría a usar en la realidad en el campo de la resistencia de materiales, para tener una herramienta que calcule una gran variedad de escenarios distintos a los que se puede someter una estructura en cuestión de segundos.

Por último, en el capítulo 7 se quiere proponer una metodología de introducción de datos en las redes, de cara a aumentar la complejidad de los problemas que podrían resolver. Para resumir, se va a realizar una lista con los objetivos del trabajo.

- Introducción al Machine Learning y a su importancia creciente en la sociedad.
- Descripción de las partes y algoritmos que componen una red neuronal.

- Descripción de la técnica de aproximación polinómica y sus similitudes y diferencias con las redes neuronales.
- Desarrollo y uso del programa Smmatlab para la generación de los datos.
- Creación, desarrollo y entrenamiento de redes neuronales capaces de resolver problemas de vigas.
- Desarrollo de una metodología de introducción de los datos en las redes neuronales para describir las estructuras y las cargas.

Capítulo 2

Redes neuronales artificiales

Las redes neuronales artificiales, ANN por sus siglas en inglés, son los algoritmos de ML más populares actualmente. Como su propio nombre indica, están inspiradas en la estructura modular del cerebro humano, con miles de neuronas interconectadas. El primer modelo que propone el uso de la neurona [1], también llamado **perceptrón**, fue el de McCulloch y Pitts en el año 1943. Este modelo de una sola neurona fue usado ampliamente, aunque el 1969 se publica el libro "Perceptrons", que pone de manifiesto sus limitaciones, por ser incapaz de clasificar datos no separables linearmente.

Fue en 1986 con el descubrimiento del algoritmo de "**backpropagation**" que se les volvió a dar importancia. Se empezaron a encadenar capas de estas neuronas para obtener modelos más complejos. No obstante, hasta está última década, y gracias al vertiginoso avance tecnológico, por el que se han multiplicado las velocidades de procesamiento y la memoria de los discos, que se ha podido ver el enorme potencial de estos modelos.

2.1. La neurona

La **neurona** es la unidad básica de procesamiento de una red neuronal [15], recibirá un número de parámetros de entrada provenientes de la capa anterior, y operándolos con sus parámetros internos, devolverá un valor de salida. En concreto, realizará una combinación lineal de las entradas sumadas a un valor de sesgo:

$$z = \mathbf{w}^T\mathbf{x} + b = \sum_{i=1}^{N} w_i \cdot x_i + b \tag{2.1.1}$$

Donde $\mathbf{x}$ es el vector de entrada a la neurona y el vector $\mathbf{w}$ y b, los parámetros internos, conocidos como pesos y sesgo respectivamente.

2.2. Funciones de activación

Por mucho que se intenten enlazar neuronas una detrás de otra, no se obtendría una mayor ventaja que con el uso de una sola neurona. Esto se debe a que la ecuación 2.1.1 es una

función lineal, y cualquier composición de funciones lineales, es una función lineal.

Es por eso que faltaría por añadir un ingrediente al comportamiento de la neurona, para garantizar que se pueda obtener un comportamiento global no lineal en la red. Este ingrediente es lo que se llama **funciones de activación** [16]. Las funciones de activación son funciones no lineales que toman como entrada la suma ponderada de la neurona, z, y devuelven la salida final o activación a. $a = f(z)$. Existen muchos ejemplos de funciones de activación, aunque las más importantes son: la unidad lineal rectificada, la tangente hiperbólica y la sigmoide.

2.2.1. Unidad lineal rectificada

También conocida como función ReLU, es una función que no realiza ninguna operación para las entradas positivas, y que devuelve el valor 0 para toda entrada negativa. Por lo tanto es una función a trozos, continua, y derivable en todo su dominio excepto en $x = 0$.

$$f(x) = \begin{cases} x, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2.2.1}$$

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases} \tag{2.2.2}$$

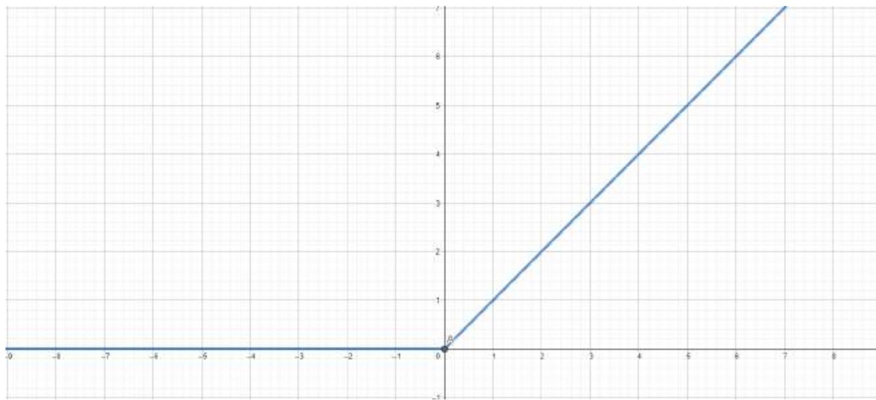


Figura 2.2.1: Función ReLU

2.2.2. Tangente hiperbólica

La salida de la función hiperbólica está acotada entre -1 y 1, y es simplemente el resultado de aplicar la función trigonométrica tangente hiperbólica.

$$f(x) = tanh(x) \tag{2.2.3}$$

$$f'(x) = \frac{1}{cosh^2(x)} \tag{2.2.4}$$

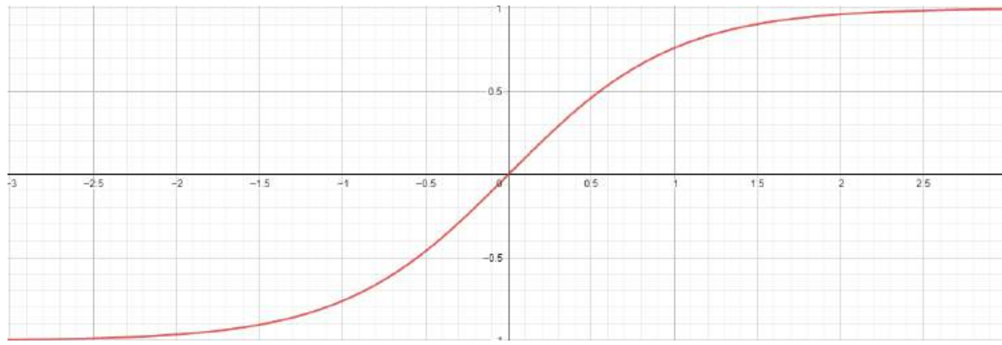


Figura 2.2.2: Tangente hiperbólica

2.2.3. Sigmoide

Es muy común en las capas de salida de las redes de clasificación, debido a que la salida se acota entre 0 y 1.

$$f(x) = \frac{1}{1+e^{-x}} \tag{2.2.5}$$

$$f'(x) = \frac{e^{-x}}{(1+e^{-x})^2} = f(x) \cdot (1 - f(x)) \tag{2.2.6}$$

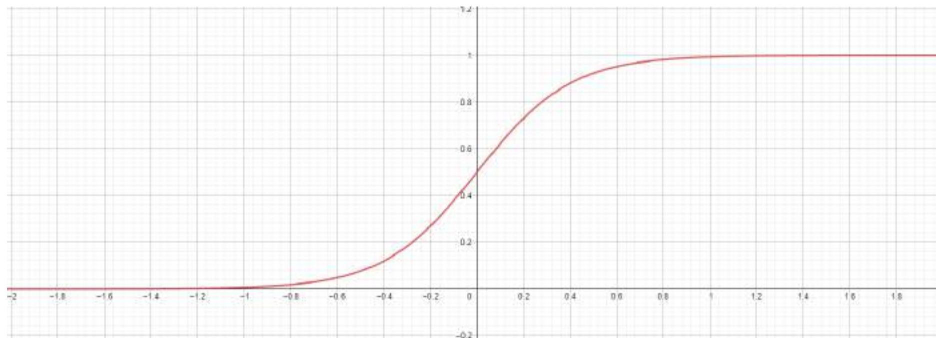


Figura 2.2.3: Función sigmoide

2.3. La red

Une vez se define el comportamiento completo de una neurona, se va a describir como se construye una red neuronal [1]. Primero se requiere de una capa de entrada, donde no se realiza ninguna operación, por lo que no se requieren neuronas, está compuesta por tantos nodos como entradas tenga la red. Saltando al otro extremo de la red, se tiene la capa de salida, compuesta por tantas neuronas como salidas, la activaciones de cada neura de la última capa corresponderán a las salidas de la red. Entre la capa de entrada y de salida se dispondrán de tantas capas como se quieran, llamadas capas ocultas, las capas ocultas podrán tener tantas neuronas como se deseen, cada neurona recibirá como entrada las activaciones

de todas las neuronas de la capa previa. En la figura 2.3.1 se muestra un esquema de una red de tres entradas y dos salidas, con dos capas ocultas de cuatro neuronas cada una.

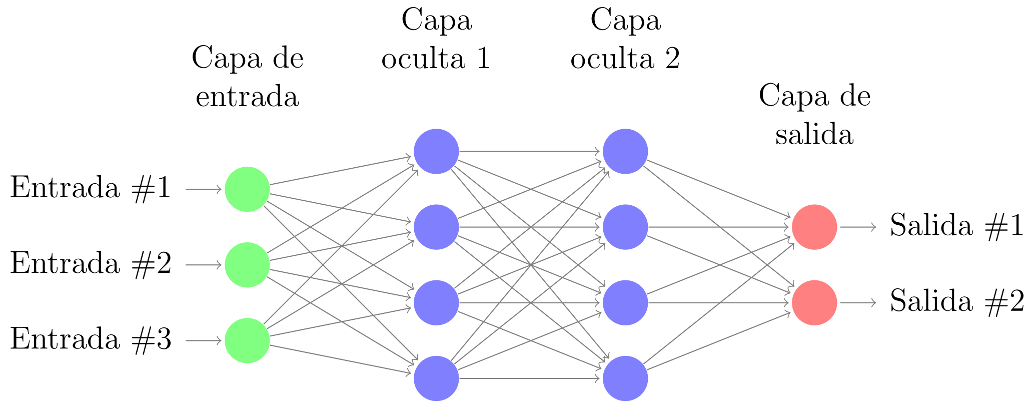


Figura 2.3.1: Esquema de una red neuronal

Además, como último componente fundamental para el funcionamiento de la red, se le debe dotar de un parámetro que estime como de bien se está ajustando a los datos propuestos. De esta manera se tiene un objetivo de aprendizaje claro, que es reducir lo máximo posible dicho parámetro. Se le conoce como **función de coste**, se pueden usar varios tipos de funciones de coste, la más común se calcula como el error cuadrático medio de las salidas estimadas por la red frente a las salidas teóricas:

$$C = \frac{1}{2N} \sum_{i=1}^{N} \sum_{j=1}^{S} (y_{ij} - \widehat{y}_{ij})^2 \tag{2.3.1}$$

Siendo N el numero de ejemplos en el dataset de entrenamiento y S la cantidad de variables de salida. En las próximas secciones se muestra como la red durante el entrenamiento busca el conjunto de parámetros **w** y b de cada neurona que **minimizan** la función de coste, a partir de métodos numéricos convencionales.

2.4. Backpropagation

Para lograr el aprendizaje de la red, será de vital importancia el cálculo de las **derivadas parciales** de la función de coste frente a los pesos y sesgos de las neuronas [17]. $\frac{\partial C}{\partial w_{jk}^{l}}$; $\frac{\partial C}{\partial b_{j}^{l}}$. Aprovechando las expresiones de las derivadas, se va a definir la notación de los subíndices que se va a emplear en el resto de la explicación. El superíndice l indica la capa dentro de la red, el subíndice j el número de neurona de dicha capa, y la k corresponde a la activación de la capa $l-1$ a la que multiplica w. Por tanto, por cada capa se puede definir una matriz $\mathbf{w^l}$ de componentes w_{jk}^{l}, con tantas filas como neuronas tenga la capa l y columnas como neuronas tenga la capa $l-1$, de la misma forma se define un vector $\mathbf{b^l}$ conteniendo todos los sesgos de la capa l. De esta forma, componiendo la función de activación con la ecuación

2.1.1 se escribe el vector de activaciones de la capa l en forma matricial.

$$\mathbf{a^l} = f(\mathbf{w^l a^{l-1}} + b^l) \tag{2.4.1}$$

Con respecto a la función de coste, se puede diseñar al gusto del programador, aunque debe cumplir dos requisitos fundamentales para poder usar el algoritmo que se va a describir aquí:

1. La función de coste de un dataset de entrenamiento de tamaño N debe ser la media aritmética de las funciones de coste de cada observación.

2. Se tiene que poder expresar en función únicamente de las activaciones de las neuronas de la última capa a_j^L

El primero de los requisitos se debe a que se van a calcular las derivadas parciales para cada observación y posteriormente se realizará la media de todas ellas para obtener la derivada parcial del dataset completo. La función de coste representada en la ecuación 2.3.1 cumple ambos requisitos y será la que se usará en los cálculos posteriores. Por tanto el problema se reduce a calcular las derivadas parciales de los pesos y el sesgo respecto de la ecuación 2.4.2, siendo L la última capa de la red.

$$C = \frac{1}{2}\sum_j (y_j - a_j^L)^2 \tag{2.4.2}$$

El algoritmo que ha hecho posible el cálculo de dichas derivadas mediante un procedimiento que sea computacionalmente viable para grandes redes se denomina 'backpropagation'. El algoritmo de backpropagation se centra primero en calcular la derivada parcial de la función de coste frente a la suma ponderada de cada neurona, lo que se denomina **error de la neurona** (Ecuación 2.4.3). El error de la neurona se interpreta matemáticamente como el cambio que sufre la función de coste con la variación de la suma ponderada que se obtiene en ella. Para su cálculo se emplea la regla de la cadena de derivación de funciones compuestas.

$$\delta_i^l = \frac{\partial C}{\partial z_i^l} \tag{2.4.3}$$

A su vez, el cálculo de los errores de las neuronas se divide en dos partes, que son, su cálculo para la última capa y su retropropagación a las capas anteriores.

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} = (a_j^L - y_j) f'(z_j^L) \tag{2.4.4}$$

$$\delta^{\mathbf{L}} = \nabla_a C \odot f'(\mathbf{z^L}) \tag{2.4.5}$$

En las ecuaciones arriba expuestas se hace patente el segundo requisito de la función de coste, y es que si dependiese de alguna variable más a parte de las activaciones de la última capa, la derivación se complicaría en exceso. El segundo de los términos de la ecuación hace referencia a la derivada de la función de activación, que podría variar según la función utilizada, pero que de todas maneras es sencillo de calcular. La ecuación 2.4.5 es la interpretación matricial de la ecuación 2.4.4, recogiendo en un único vector los errores de las neuronas de la última

capa al completo, y siendo el producto $\odot$ la multiplicación matricial elemento a elemento.

Para el cálculo de los errores de las neuronas de capas previas se parte del error en la capa posterior, $\delta_k^{l+1} = \frac{\partial C}{\partial z_k^{l+1}}$.

$$\delta_j^l = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial a_j^l} \frac{\partial a_j^l}{\partial z_j^l} = \sum_k \delta_k^{l+1} w_{kj}^{l+1} f'(z_j^l) \tag{2.4.6}$$

$$\delta^{\mathbf{l}} = ((\mathbf{w}^{\mathbf{l+1}})^T \delta^{\mathbf{l+1}}) \odot f'(\mathbf{z}^{\mathbf{L}}) \tag{2.4.7}$$

Igual que antes, las ecuaciones 2.4.6 y 2.4.7 son equivalentes, siendo la segunda su representación matricial. El término $\frac{\partial z_k^{l+1}}{\partial a_j^l}$ se obtiene de derivar la ecuación 2.1.1 respecto de las activaciones de las capas anteriores. De esta manera se aprecia como se pueden ir obteniendo los errores de las neuronas de todas las capas a partir de los errores de las capas posteriores y su matriz de pesos.

La segunda parte del algoritmo consiste en relacionar los errores de las neuronas con las derivadas parciales del coste respecto de los pesos y los sesgos, lo que se obtiene derivando la ecuación 2.1.1.

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \tag{2.4.8}$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \cdot 1 \tag{2.4.9}$$

Llegados a este punto ya se cuentan con todas las herramientas necesarias para calcular las derivadas parciales de la función de coste respecto a los parámetros que se modificarán durante el entrenamiento. La idea esencial de este algoritmo es que se parte de un dataset de entrenamiento, con N parejas de vectores $\mathbf{x_i}$ e $\mathbf{y_i}$. Por cada pareja se hace el cálculo hacia adelante, partiendo de $\mathbf{x_i}$ se obtienen: $\mathbf{z_i^2}$, $\mathbf{a_i^2}$ ..., $\mathbf{z_i^L}$, $\mathbf{a_i^L}$ y C. Después se prosigue hacia atrás partiendo de la función de coste: $\delta_\mathbf{i}^\mathbf{L}$, $\frac{\partial C_i}{\partial \mathbf{w_{jk}^L}}$, $\frac{\partial C_i}{\partial \mathbf{b_j^L}}$, ..., $\delta_\mathbf{i}^\mathbf{2}$, $\frac{\partial C_i}{\partial \mathbf{w_{jk}^2}}$, $\frac{\partial C_i}{\partial \mathbf{b_j^2}}$. Una vez obtenidas todas las derivadas parciales necesarias se realiza la media aritmética sobre todo el dataset, $\frac{\partial C}{\partial w_{jk}^l} = \frac{1}{N} \sum_{i=1}^N \frac{\partial C_i}{\partial \mathbf{w_{jk}^l}}$, con lo que se obtiene el gradiente del coste respecto de los parámetros de la red.

$$\nabla_{\mathbf{w},\mathbf{b}} C \tag{2.4.10}$$

2.5. Descenso del gradiente

El entrenamiento de una red neuronal comienza con una inicialización aleatoria de los parámetros de la red, posteriormente se seguirá un proceso iterativo del cálculo del gradiente y actualización de los parámetros. El descenso del gradiente definirá como se usará el gradiente para **actualizar los parámetros** de la red. Como ya se ha comentado, el objetivo del aprendizaje es reducir al máximo el coste, por tanto, se buscará encontrar un mínimo local

o global de la función de coste. La forma clásica de calcular el mínimo de una función es sacar el gradiente, igualarlo a cero y resolver el sistema. $\nabla f = \mathbf{0}$, sin embargo, la falta de una expresión matemática concreta dependiente de los pesos y sesgos de la función de coste, imposibilita usar este método.

No obstante, gracias al algoritmo de backpropagation se cuentan con todas las derivadas parciales, así que para cada peso de cada neurona, se tiene una ligera idea de como varia el coste al modificar dicho parámetro, por tanto, si la derivada parcial es positiva, se sabe que un incremento del peso supondrá un aumento del coste [17]. Como se desea todo lo contrario, una derivada parcial positiva servirá para que el peso se actualice a valores mas bajos y una negativa hacia valores más altos. Además, la magnitud también proporciona pistas de la brusquedad con la que modifica el coste, un valor elevado requiere un cambio del peso elevado, y un valor pequeño indica proximidad al punto estacionario, por lo que la actualización será más sutil. De esta manera se puede iterar el valor de cada peso y sesgo hasta alcanzar el coste deseado:

$$w_{n+1} = w_n - \alpha \frac{\partial C}{\partial w_n} \tag{2.5.1}$$

En la ecuación 2.5.1 se muestra la fórmula del descenso del gradiente para la actualización de un peso cualquiera de la red. Durante el entrenamiento, se aplicará la ecuación a todos los parámetros de la red en cada iteración. El parámetro α se conoce como **tasa de aprendizaje** y está relacionado con la velocidad de convergencia hacia el mínimo de la función de coste. Una tasa de aprendizaje baja implica un aprendizaje lento, pues se requerirán muchas iteraciones para encontrar un mínimo. Por otro lado, una tasa de aprendizaje elevada implica una mayor inestabilidad en el algoritmo, que da saltos mucho más grandes, lo que puede significar que nunca se alcance la convergencia. Una de las soluciones usadas para buscar un compromiso entre velocidad de convergencia y estabilidad es usar una tasa de aprendizaje elevada en las primeras iteraciones e ir reduciéndola paulatinamente según avanzan las iteraciones. Una forma más global de escribir la ecuación 2.5.1 es usando el gradiente de la ecuación 2.4.10.

$$[\mathbf{w}, \mathbf{b}]_{n+1} = [\mathbf{w}, \mathbf{b}]_n - \alpha \cdot \nabla_{\mathbf{w},\mathbf{b}} C_n \tag{2.5.2}$$

2.6. Modificaciones al descenso del gradiente clásico

En la segunda parte del documento, durante la programación, se utilizará la librería de Matlab 'Deep Learning Toolbox' [18]. En dicha librería se ofrecen tres algoritmos de entrenamiento basados en el descenso del gradiente, a los que se les denominan solvers, los solvers modifican el descenso del gradiente para mejorar la estabilidad y la convergencia.

2.6.1. Stochastic Gradient Descent with Momentum (SGDM)

En español, descenso del gradiente estocástico con momento, trata de reducir la oscilación para aumentar la estabilidad. Usa un parámetro γ para considerar la contribución de los

parámetros de la iteración previa.

$$[\mathbf{w}, \mathbf{b}]_{n+1} = [\mathbf{w}, \mathbf{b}]_n - \alpha \cdot \nabla_{\mathbf{w},\mathbf{b}} C_n + \gamma \cdot ([\mathbf{w}, \mathbf{b}]_n - [\mathbf{w}, \mathbf{b}]_{n-1}) \tag{2.6.1}$$

2.6.2. RMSProp

Mientras que SGDM usa la misma tasa de aprendizaje para todos los parámetros, RMSProp calcula una media móvil (ecuación 2.6.2) de los cuadrados de los gradientes, con el objetivo de usar una tasa más pequeña para los parámetros con grandes gradientes y una más grande para los pequeños gradientes y que la actualización sea más uniforme.

$$\nu_{\mathbf{n}} = \beta_2 \nu_{\mathbf{n-1}} + (1 - \beta_2)(\nabla_{\mathbf{w},\mathbf{b}} C_n)^2 \tag{2.6.2}$$

El parámetro β_2 se le conoce como 'decay rate', que significa algo así como tasa de decrecimiento, y suele oscilar en torno a 0.99. La ecuación de actualización del solver RMSProp se puede escribir como:

$$[\mathbf{w}, \mathbf{b}]_{n+1} = [\mathbf{w}, \mathbf{b}]_n - \frac{\alpha \cdot \nabla_{\mathbf{w},\mathbf{b}} C_n}{\sqrt{\nu_{\mathbf{n}}} + \epsilon} \tag{2.6.3}$$

Siendo la división elemento a elemento y ϵ una constante pequeña para evitar la división entre 0.

2.6.3. Adam

El último solver propone una solución que aglutina las dos anteriores. Por un lado usa ν para proporcionar distintas tasas de aprendizaje para cada parámetro. Por otro lado incluye un termino de momento, que se calcula como la media móvil de los gradientes, y sirve para reducir el ruido.

$$\mathbf{m_n} = \beta_1 \mathbf{m_{n-1}} + (1 - \beta_1) \nabla_{\mathbf{w},\mathbf{b}} C_n \tag{2.6.4}$$

$$\nu_{\mathbf{n}} = \beta_2 \nu_{\mathbf{n-1}} + (1 - \beta_2)(\nabla_{\mathbf{w},\mathbf{b}} C_n)^2 \tag{2.6.5}$$

$$[\mathbf{w}, \mathbf{b}]_{n+1} = [\mathbf{w}, \mathbf{b}]_n - \frac{\alpha \cdot \mathbf{m_n}}{\sqrt{\nu_{\mathbf{n}}} + \epsilon} \tag{2.6.6}$$

2.7. Levenberg-Marquardt

El descenso del gradiente no es el único método disponible para minimizar la función de coste en un red neuronal, de hecho, hay muchos más. Por su sorprendente buen funcionamiento con los problemas que se va a tratar, se explica el algoritmo de Levenberg-Marquardt.

Este método es una combinación del descenso del gradiente y el **método de Newton** [19]. En el método de Newton, para aumentar la velocidad de convergencia, se usan las segundas

derivadas de la función en forma de la matriz Hessiana. No obstante, como ya se ha explicado antes, el algoritmo de backpropagation solo proporciona información sobre la primera derivada, siendo computacionalmente muy costoso el cálculo de la segunda. Para solucionar dicho problema, existe una forma de aproximar su valor para funciones que tienen la forma de suma de cuadrados, como es el caso del error cuadrático medio (Ecuación 2.7.1).

$$\mathbf{H} = \mathbf{J}^{\mathbf{T}}\mathbf{J} \tag{2.7.1}$$

La ecuación de actualización del método de Levenberg-Marquardt (Ecuación 2.7.2) cuenta con un parámetro, μ. Cuando μ toma valores grandes, la matriz Hessiana pierde peso en la ecuación, y su funcionamiento será similar al descenso del gradiente con valores pequeños de α. En cambio, cuando μ sea pequeño, la matriz Hessiana gana importancia, y su comportamiento es similar al método de Newton.

$$[\mathbf{w}, \mathbf{b}]_{n+1} = [\mathbf{w}, \mathbf{b}]_n - [\mathbf{J}^{\mathbf{T}}\mathbf{J} + \mu\mathbf{I}]^{-1}\nabla_{\mathbf{w},\mathbf{b}}C_n \tag{2.7.2}$$

Para terminar con la explicación del método, añadir que existe una estrategia para ir variando μ durante el entrenamiento y optimizar el proceso. Tras cada epoch, se compara el coste final con el de la epoch anterior, si el coste ha subido y la red no esta aprendiendo, μ se multiplica por un factor mayor que uno, por tanto crece, para acercarse al comportamiento del descenso del gradiente y avanzar con mayor fiabilidad. Por el contrario, cuando el coste baje, μ se multiplica por una factor menor que uno, para aumentar la velocidad de convergencia.

2.8. Regularización

Uno de los grandes problemas de las redes neuronales, y en general, de cualquier modelo complejo con un gran número de parámetros, se refiere su capacidad de generalizar con observaciones con los que no han sido entrenados. Este problema es conocido como overfitting [20] y surge de que en muchas ocasiones, la red se limita a "aprenderse de memoria" las observaciones de entrenamiento, y posteriormente, al enfrentarse a problemas nuevos, su rendimiento es deficiente, debido a que no ha sido capaz de captar las relaciones subyacentes entre las entradas y las salidas [17]. En grandes redes neuronales, donde el número de pesos de la red es inmenso, es muy común el surgimiento de este problema.

Para solventar este problema, lo primero que se debe hacer es usar modelos con un complejidad adecuada para el problema en cuestión, y en equilibrio con el tamaño del dataset de entrenamiento. No obstante, existe otro método que limitará la aparición de overfitting. El método se conoce como regularización, es ampliamente utilizado en modelos estadísticos y en redes neuronales es común usar la regularización tipo L2.

Su implementación consiste en añadir un término extra en la función de coste dependiente de los pesos.

$$C = C_0 + \frac{\lambda}{2}\sum_{w} w^2 \tag{2.8.1}$$

EL parámetro λ es el parámetro de regularización y toma un valor mayor que cero. Cuanto mayor sea λ, mayor preferencia se le dará a reducir el valor de los pesos sobre reducir C_0 y viceversa. De primeras, no es obvio porque este método reduce la aparición de overfitting, pero experimentalmente funciona. De manera intuitiva, se puede explicar diciendo que una red en el que los pesos no puedan crecer indefinidamente, manteniéndose en valores cercanos a cero, tendrá una menor complejidad, y modelos de menor complejidad tienen menor capacidad para aprenderse de memoria los datos.

2.9. Teorema de aproximación universal

Para finalizar el capítulo, y cambiando un poco de tercio, se va a intentar contestar a una pregunta de vital importancia para comprender la utilidad y potencia de las redes neuronales. ¿Pueden las redes neuronales aprender a comportarse como cualquier función matemática? [21] La respuesta a esta pregunta es que si, pero con matices. El primero de los matices es que sería más correcto decir que pueden aproximar cualquier función matemática con toda la precisión que se desee. Es decir, si tenemos una función $f(x)$ cualquiera que es aproximada por una red neuronal $g(x)$, siempre podremos encontrar una red tal que $|f(x) - g(x)| < \epsilon$, para cualquier ϵ que queramos. El segundo de los matices tiene que ver con la continuidad de la función, y es que funciones discontinuas con saltos repentinos no serán posibles de aproximar en muchos casos, debido a que las funciones calculadas por redes neuronales son continuas.

La primera vez que se demuestra que las redes neuronales pueden aproximar cualquier función es en Hornik, 1991. Además, en el capitulo 4 del libro digital 'Neural networks and deep learning' [17], se demuestra de una forma sencilla y visual. Para ello usa redes de una sola capa oculta para funciones con una entrada, y con dos capas ocultas para funciones de más de una entrada, aunque asegura que también es posible hacerlo con una sola capa. La demostración consiste en utilizar funciones de activación de tipo escalón para obtener una función de salida compuesta por escalones de distintos anchos y alturas, que con la incorporación de un suficiente número de neuronas y la manipulación correcta de los pesos, se pueden alterar para tener el ancho y la altura deseados así como moverlos por el eje X. En la figura 2.9.1 se puede observar como se aproxima una función arbitraria con dichos escalones.

Algo similar se podría realizar usando funciones de activación tipo unidad lineal rectificada, obteniendo funciones lineales continuas a trozos, similares a las que se usan con el método de los elementos finitos. El número de elementos que puede usar la red neuronal para aproximar dependerá directamente del número de neuronas, por lo que se puede decir que, al igual que en el método de los elementos finitos la precisión en general aumenta conforme se usan más elementos, la precisión de una red neuronal aumenta conforme se usan más neuronas.

Las redes neuronales no son el único aproximador universal que existe. Uno de los más usados, por su simplicidad, son las funciones polinómicas. Posteriormente se usará para los problemas ya comentados, con la intención de comparar su rendimiento con el de las redes neuronales.

2.9.1. Deep learning

En los últimos años las redes neuronales están creciendo en profundidad, cada vez contando con más capas ocultas, es natural preguntarse por qué el interés en hacer crecer estas redes si en el párrafo anterior se afirmaba que el teorema de aproximación universal era valido para redes de una sola capa. La respuesta a esta pregunta tiene en parte que ver con características emergentes y comportamientos inteligentes que surgen del aprendizaje de las redes profundas. Por ejemplo, en una red convolucional de clasificación de imágenes se observa que en las primeras capas la red tiene comportamientos sencillos y concretos, como identificar bordes o patrones rayados, en cambio, en capas más profundas el aprendizaje se vuelve más abstracto, pudiendo reconocer texturas, paisajes, objetos o personas. Esto brinda la oportunidad de usar las redes para una tarea completamente distinta, como puede ser el de generar diseños de un objeto concreto.

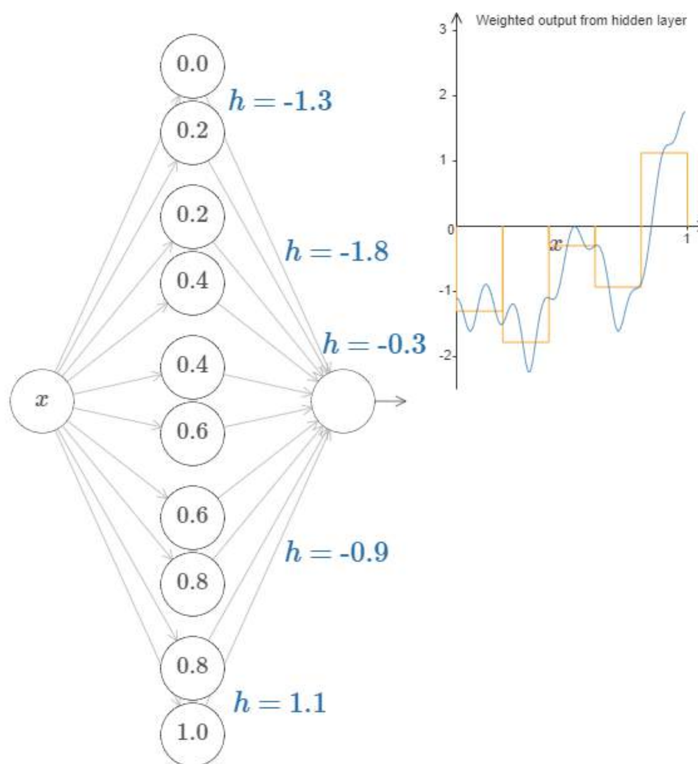


Figura 2.9.1: Aproximación de una función arbitraria por una red neuronal. Extraída de 'Neural networks and deep learning' [17]

Capítulo 3

smMatlab y generación de datasets

3.1. smMatlab

smMatlab es un programa de resolución de problemas de vigas que se ha hecho con la intención de resultar útil desde el punto de vista educativo. Está programado en Matlab, que es el lenguage con el que se sienten más cómodos los alumnos de 3º de la ETSII, año en el que se imparte la asignatura *Resistencia de Materiales*, asignatura para la que está pensado el programa.
Hay numerosos códigos de resolución de vigas comerciales e incluso en internet para pequeños problemas. Sin embargo, con smMatlab se pretende dar un enfoque educativo para dar los primeros pasos tanto en la Resistencia de Materiales como en la mecánica computacional. El programa en sí es una colección de clases definidas que hacen referencia a diferentes conceptos físicos que aparecen en el estudio de la Resistencia de Materiales y que permiten modelar y resolver problemas de vigas. El concepto de clase es el fundamento del programa y de su finalidad, pues con ella se pretende definir computacionalmente conceptos físicos elementales de la Resistencia de Materiales. Con esto se pretende por un lado reforzar el conocimiento de la materia, pues enfrentarse al problema de la definición de conceptos para el ordenador requiere un entendimiento sustancial de la materia (recordemos la definición de Feynman de ordenador). Por otro lado, acostumbrar al alumno con la programación en general (y orientada a objetos en particular), la cual es fundamental para el mundo actual.
Conviene recalcar aquí, que dado el carácter educativo del programa, se ha primado la claridad y la resolución análoga a la hecha *a mano* en lugar de la velocidad de cómputo. No se pretende con este programa calcular ninguna estructura real, sino introducir al alumno en los conceptos básicos a través de la computación. Así pues, el programa está pensado con cálculo simbólico, el cual es visualmente muy parecido a lo escrito por un alumno y, por otro lado, mucho más lento que el uso de matrices numéricas, que podrían utilizarse en lugar de las ecuaciones simbólicas que se obtienen.
El flujo del programa es el siguiente

1. Crear las vigas (ver 3.1.3), las conexiones entre ellas (ver 3.1.2) y los apoyos (ver 3.1.1) que forman la estructura.

2. Crear la estructura (ver 3.1.4).

3. Definir las fuerzas externas que actúan sobre la estructura.
4. Definir el problema creando un objeto de tipo `smProblem` (Ver 3.1.5). Al crear este objeto, se hace lo siguiente:
    - Se almacena la estructura (ver 3.1.4), donde se describe el conjunto de vigas, nodos y apoyos que definen la estructura real.
    - Se relacionan las cargas con las vigas o nodos en las que se aplican.
5. Una vez creado y almacenado el problema, para resolver el problema únicamente hay que llamar al método `solve(obj)` con el que se crea el sistema de ecuaciones (ya sea por equilibrio y compatibilidad geométrica o por minimización de la energía). Para ello se llama a la estructura, que a su vez llama a todas las vigas y nodos de la que está compuesta para obtener las ecuaciones. Una vez obtenidas las ecuaciones, se resuelve el problema con los métodos disponibles en Matlab (principalmente `linsolve()` y `fmincon()`).

Se describen a continuación las clases principales del programa.

3.1.1. Clase `support`

Es la clase que define los apoyos. Recordemos que un apoyo es un elemento que impide cierto movimiento, ya sea traslacional o rotacional. Como consecuencia de esta restricción surge una carga (fuerza o momento) en dirección del movimiento restringido, que debe tenerse en cuenta en el cálculo de la estructura. Su definición debe hacerse, por tanto, en primer lugar con la posición que ocupa y en segundo lugar en base a las direcciones de los movimientos que impide y al tipo (traslacional o rotacional) que impide.
Existen por supuesto algunos apoyos que se repiten en numerosas ocasiones y que conviene tener definidos como subclases de la clase `support`. Por ejemplo:

- `fixed`. Modela lo que físicamente es un empotramiento y restringe todos los movimientos, tanto traslacionales como rotacionales.
- `pinned`. Modela lo que físicamente es un apoyo simplemente apoyado y restringe todos los movimientos traslacionales, permitiendo los rotacionales.
- `roller1` y `roller2`. Modela lo que físicamente son apoyos simplemente apoyados, permitiendo el movimiento en 1 o 2 direcciones, respectivamente. En este caso se restringe las traslaciones en el espacio ortogonal a los movimientos permitidos (al plano perpendicular en el caso de 1 dirección permitida o a la recta perpendicular al plano definido por las dos direcciones permitidas en el caso de 2 direcciones). Como se puede ver, en este caso es más sencillo construir el objeto a partir de las direcciones que se permiten, pues generalmente es lo que se conoce. Por tanto, la definición de las clases asociados con estos apoyos se construyen a partir de las direcciones permitidas e internamente se calculan las restringidas.

Por supuesto, se puede definir cualquier tipo de apoyo a partir de la clase genérica `support`.

3.1.2. Clase `connection`

La clase `connection` define los *enlaces* entre vigas. Recordemos que un *enlace* entre vigas permite cierto movimientos (traslacional o rotacional) entre los extremos de las vigas que une. Como consecuencia de ello, el esfuerzo interno en la dirección del movimiento permitido en el enlace debe ser nulo, lo cual debe tenerse en cuenta para el cálculo de la estructura.
En general, un enlace puede unir varias vigas en un solo punto. Sin embargo, estos enlaces no tienen que ser igual entre todas las vigas. Por ejemplo, puede ser que tres vigas que se unan físicamente en un punto y dos de ellas estén unidas mediante una rótula (permite las rotaciones) y la otra tenga una unión rígida (no se permite ningún movimiento). Por tanto, la definición de un enlace genérico debe tener en cuenta el tipo de enlace que hay entre cada par de vigas. Este tipo de *enlace simple* entre dos barras está definido en smMatlab con la clase `basicConnection`. Y como se ha explicado, queda definido por el par de vigas que une y los movimientos permitidos entre ellas. Como en el caso de los apoyos, algunos enlaces simples se repiten en numerosos problemas y están definidos a través de subclases de `basicConnection`:

- `cBjoint`. Modela lo que físicamente es una rótula esférica y permite todas las rotaciones entre los extremos de las vigas que une.
- `cHinge`. Modela lo que físicamente es una rótula que permite las rotaciones en el plano de la sección, siendo el ángulo de torsión el mismo en ambos extremos.
- `cRigid`. Modela una conexión rígida. Ningún movimiento entre los extremos de viga está permitido.

Por último, la definición del *enlace* en el caso general, `connection`, se hace a través de un conjunto de `basicConnection`.

3.1.3. Clase `beam`

La clase principal del programa es la clase `beam`. Como su nombre indica, es la clase en la que se encuentra la propia de definición de viga. Recordemos que una viga es un modelo simplificado en la mecánica del sólido deformable que pretende modelar los elementos constructivos que llamamos vigas y que no son más que elementos con una dimensión principal mucho mayor que las otras dos.
Formalmente la viga se define geométricamente como el producto de dos variedades $\mathcal{B} = \mathcal{C} \times \mathcal{S}$, donde $\mathcal{C}$ es la *variedad base*, definida por una curva, y $\mathcal{S}$ es la *fibra*, definida por las secciones rectas a los puntos de la curva $\mathcal{S}$ (REF: Epstein). Recordemos además, que por conveniencia, se define la curva $\mathcal{C}$ como la que une los baricentros de las secciones $\mathcal{S}$. Y además se define un sistema de referencia $Gxyz$ centrado en el baricentro de cada sección, G, en el que el eje x es perpendicular a la sección y los ejes y, z son ejes principales de inercia en la sección. Esta definición es importante, ya que indica los dos primeros elementos que necesariamente tienen que aparecer en la definición de la viga en la clase `beam`, es decir, la curva de los baricentros ($\mathcal{C}$) y las secciones para cada punto de la curva.
Además, la viga deformable necesita definir un material constitutivo (y su ley) para completar el modelo físico necesario. Dependiendo del tipo de ley constitutiva (isótropo elástico lineal,

elástico no lineal, plástico...) y de otras hipótesis, el modelo de viga será uno u otro. En la asignatura *Resistencia de Materiales* únicamente se estudia el caso de materiales continuos, isótropos, elásticos lineales y además se hace la hipótesis de despreciar el cortante (viga de Euler-Bernoulli), que es la que está programada en smMatlab. Pero es importante recalcar que otro tipo de viga (por ejemplo, la viga de Timoshenko) puede programarse como subclase de la clase que define geométricamente la viga.
Por tanto, los parámetros fundamentales de la clase viga son los que definen la curva $\mathcal{C}$ (`dirFun`, `dirFunInv`, `dirFunDxds`, `dirAxi` y `divVar`), los que definen las secciones $\mathcal{S}$ (`secFun`, `secVar`). Y para el caso de la viga de Bernoulli se define el material isótropo, elástico y lineal. El resto de variables definidas son simplemente para realizar cálculos.
A partir de aquí, la clase viga tiene numerosos métodos a los que se puede llamar, destacando especialmente los siguientes por su importancia en la resolución de problemas:

- `getLoadsEqns(obj,ecF,ecM)`. Es el método encargado de calcular las ecuaciones de equilibrio en la viga a partir de las cargas externas `ecF` (fuerzas) y `ecM` (momentos).

- `getDisplacementsEqns(obj)`. Es el método encargado de relacionar el desplazamiento entre el principio y el final de la viga a partir de las deformaciones.

- `setDeformationLaw(obj)`. Es el método encargado de establecer las leyes de deformaciones en la viga, es decir, calcula la derivada de los desplazamientos y de las rotaciones respecto al parámetro de la viga.

- `setIntFLaw(obj,ecF,ecM)`. Es el método encargado de calcular las leyes de esfuerzos internos (normal, cortantes, torsor y flectores) a partir de las cargas externas `ecF` (fuerzas) y `ecM` (momentos) , que deben darse en coordenadas globales.

3.1.4. Clase `structure`

La clase `structure` tiene como finalidad almacenar el conjunto de la estructura que se analiza. Es la definición computacional de una estructura física. Para ello guarda el conjunto de vigas que forman el problema (`beams`), el conjunto de nodos, que son los extremos de las vigas (`nodes`, el conjunto de apoyos (`supports`) y finalmente la conectividad, es decir la relación entre los nodos y las vigas.
Los tres métodos principales de esta clase son:

- `getDisplacementsEqns(obj)`. Recorre todas las vigas y nodos en la estructura y obtiene las ecuaciones de compatibilidad geométrica correspondientes.

- `getEnergy(obj)`. Recorre todas las vigas de la estructura para obtener la energía almacenada.

- `getLoadsEqns(obj,ecFb,ecMb,ecFn,ecMn)`. Recorre todas las vigas y nodos de la estructura y obtiene las ecuaciones de equilibrio correspondientes.

3.1.5. Clase `smProblem`

La clase smProblem es la clase en la que se guardan todos los datos del problema. Cuando se resuelve un problema con smMatlab, lo primero que se hace es crear un objeto de tipo smProblem. Es la clase equivalente a la definición del problema propiamente dicho, al enunciado. En este objeto se almacena:

- `structure`. Este tipo de clase guarda la estructura con toda su geometría y su conectividad. Ver ejemplo para más detalle.
- Las condiciones de frontera del problema. Tando las fuerzas externas aplicadas (`bpFloads`, `bdFloads`, `bpMloads`, `bdMloads`), como las fuerzas debidas a los apoyos (`ueF`, `ueM`).
- El resto de variables sirven para facilitar los cálculos.

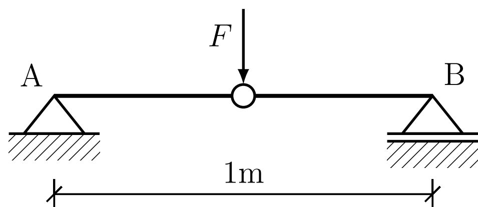
Dentro de la clase `smProblem` el método más relevante es `solve()`, con el que se resuelve el problema. La resolución se puede realizar de dos formas diferentes:

- Resolviendo el sistema de ecuaciones de equilibrio en las vigas y en los nodos, así como la compatibilidad geométrica por las deformaciones.
- Minimizando la energía potencial, $V(\boldsymbol{u}, \boldsymbol{r}) = W(\boldsymbol{u}, \boldsymbol{r}) - U(\boldsymbol{u}, \boldsymbol{r})$

Tradicionalmente en la asignatura *Resistencia de Materiales* se sigue la primera opción o quizás una mezcla utilizando el teorema de Castigliano (que se puede ver como una consecuencia de la minimización de la energía potencial). Con la finalidad de obtener cierta analogía en el método `solve` se obtienen siempre las ecuaciones de equilibrio en todas las vigas y nodos. En el caso de que el problema sea hiperestático se obtienen entonces las ecuaciones de compatibilidad geomética en los nodos o se minimiza la energía potencial, dependiendo del método elegido para resolver el problema (`solverType`).

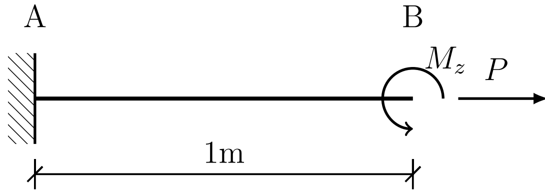
3.1.6. Ejemplos

Para ilustrar el funcionamiento del programa, y que se entienda su uso con mayor claridad, se incluyen dos archivos de código de Matlab a modo de ejemplo. Dentro del código se siguen los pasos explicados previamente, creando vigas, apoyos, conexiones, estructura, fuerzas externas y por último el objeto smProblem, complementado con comentarios en español describiendo la información necesaria que se necesita aportar. El primer problema se trata de una viga simplemente apoyada de un metro de longitud con una rótula en la parte central y una carga vertical en la rotula.



```
% Ejemplo de uso de smmatlab: Viga simplemente apoyada con rotula
% en el medio


% definición de las vigas, se definen dos vigas adyacentes de medio
% metro de longitud, bm1 y bm2. Para ello se usan los puntos del
% espacio cartesiano de comienzo y fin de las vigas.

bm1=csBeam([[0.0 0.0 0.0]' [0.5 0.0 0.0]']);
bm2=csBeam([[0.5 0.0 0.0]' [1.0 0.0 0.0]']);

% Conexión entre las vigas, se utiliza la subclase cHinge, que
% define el comportamiento de una rotula, y se declara la variable
% cn1, que modela la conexión de la rotula entre dos vigas.

bcn1 = cHinge();

cn1 = connection(2,{bcn1});

% Se define un apoyo empotrado, s1, en el punto [0,0,0] y una apoyo
% móvil, s2, en dirección de la viga (Eje X), en el punto [1,0,0]

s1=fixed([0;0;0]);
s2=roller1([1;0;0],[1;0;0]);

% Definición de la estructura, se introducen las vigas bm1 y bm2, la
% conexión, la cual se posiciona en el punto [0.5, 0, 0], y entre
% las vigas bm1 y bm2, y los apoyos s1 y s2.

s=structure({bm1,bm2},{{cn1,[0.5;0.0;0.0],{bm1,bm2}}},{s1,s2});

% Fuerzas externas, se aplica una fuerza F = [0, -1, 0] N, en el
% punto [0.5, 0, 0]

bF=[0.5;0.0;0.0;0.0;-1.0;0.0];

% Se define el preblema como unión de la estructura y de la carga
sm=smProblem(s,bF);

% Resolución del problema

sm.solve();
```

En el segundo problema de ejemplo se define una viga de un metro de longitud y sección circular constante de 20 cm de radio empotrada en un extremo. Se define además un material isótropo con modulo de Young y coeficiente de Poisson distintos a los del acero. En cuanto a las cargas, se aplicará un fuerza axial y un momento en el eje Z en el extremo no empotrado.



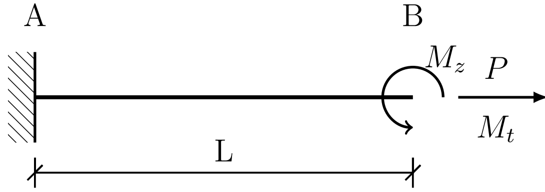

```
% Ejemplo de uso de smmatlab: Viga empotrada de sección circular
% y material isótropo con cargas en el extremo

% Definición de la viga: Longitud (L), y material, E y nu
L = 1;
E = 70;
nu = 0.28;
bm = csBeam([[0.0 0.0 0.0]' [L 0.0 0.0]'], {'Mat', [8e3, E*1e9, nu]}, ...
            {@cirSec, {20e-2}});

% Definición de las cargas, bF, carga axial en el extremo final, bM,
% momento en el extremo final, bempty, se trata de una fuerza
% distribuida de valor cero necesaria en la declaración de la
% clase smProblem
P = 5000;
bF=[L;0.0;0.0;P;0.0;0.0];
Mz = 3000;
bM=[L;0.0;0.0;0;0.0;Mz];
bempty = [0.0;0.0;0.0;0.0;0.0;0.0];

% Definición del apoyo empotrado, subclase fixed, en el extremo
% inicial
s1 =  fixed([0.0;0.0;0.0]);

% Declaración de la estructura y el problema, así como su resolución
% con el método solve
s=structure({bm},{},{s1});
sm=smProblem(s,bF,bempty,bM);
sm.solve
```

3.2. Viga con esfuerzos constantes

El primero de los problemas que se van a a utilizar consiste en una viga circular de 20 cm de radio de un material arbitrario, empotrada en un extremo y sometida a tres cargas en el otro: Fuerza axial, momento flector y momento torsor. Esta disposición va a generar una distribución de esfuerzos constantes en toda la viga, y deformación en sentido vertical, axial y giro de torsión. Las variables que se van a usar como entradas del problema y el rango numérico de estás son:

1. Longitud, $L : 4 - 20m$
2. Fuerza axial, $P : 2000 - 10000N$
3. Momento flector, $M_z : 2000 - 10000Nm$

4. Momento torsor, $M_x : 2000 - 10000Nm$
5. Modulo de Young, $E : 60 - 210GPa$
6. Modulo de elasticidad transversal, $G : 22 - 84GPa$



Para la generación de los datasets se empleará un script de Matlab, código 1 del apéndice A, los datos se generarán de forma aleatoria siguiendo una distribución uniforme con la función `rand` de Matlab. Se consideran soluciones a este problema los esfuerzos normal, flector y cortante, y las deformaciones longitudinales y angulares. Las relaciones matemáticas entre las entradas y las salidas son:

1. Esfuerzo normal, $N = P$
2. Esfuerzo flector, $M_f = M_z$
3. Esfuerzo torsor, $M_t = M_x$
4. Deformación axial, $u(x) = \frac{Nx}{EA}$
5. Deformación vertical, $v(x) = \frac{M_z x^2}{EI_z}$
6. Ángulo de flexión, $\theta(x) = \frac{2M_z x}{EI_z}$
7. Ángulo de torsión, $\theta_x(x) = \frac{M_x L}{GI_0}$

En la tabla 3.1 se incluye un porción del dataset generado, con el propósito de homogeneizar los ordenes de magnitud de las entradas y salidas para facilitar el entrenamiento, se realizan algunos cambios de unidades, reflejados en la tabla.

Entradas						
L(m)	P(kN)	$M_x(kNm)$	$M_z(kNm)$	$E(GPa)$	$G(GPa)$	
16.009	8.6816	4.3839	3.7869	164.6	65.3912	
5.3188	7.1232	4.1288	5.1832	104.5778	39.3391	
10.9365	7.0535	7.5692	3.937	113.5495	45.2781	
9.6292	3.1614	9.2621	7.8558	70.6234	27.5492	
8.7546	6.1446	5.0289	5.0052	117.0992	44.1	
11.3649	3.1383	5.2965	6.2894	114.3056	43.1071	

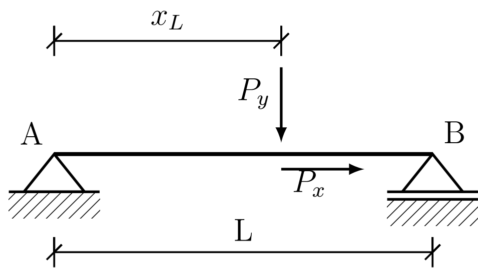
Salidas												
$N(kN)$	$M_t(kNm)$	$M_f(kNm)$	$v(0.25L)(nm)$	$\theta \cdot 10^9(0.25L)$	$v(0.5L)(nm)$	$\theta \cdot 10^9(0.5L)$	$v(0.5L)(nm)$	$\theta \cdot 10^9(0.75L)$	$v(L)(nm)$	$\theta \cdot 10^9(L)$	$u(L)(nm)$	$\theta_x \cdot 10^7$
8.6816	4.3839	3.7869	1.4663	7.3274	5.8652	14.6548	13.1967	21.9822	23.4608	29.3096	6.7193	4.2703
7.1232	4.1288	5.1832	0.3487	5.2444	1.3947	10.4889	3.1381	15.7333	5.5788	20.9777	2.883	2.2211
7.0535	7.5692	3.937	1.0313	7.5437	4.1251	15.0875	9.2815	22.6312	16.5004	30.1749	5.4061	7.2745
3.1614	9.2621	7.8558	2.5648	21.3088	10.2593	42.6177	23.0834	63.9265	41.0372	85.2353	3.4301	12.8809
6.1446	5.0289	5.0052	0.8147	7.4444	3.2586	14.8888	7.3319	22.3332	13.0345	29.7775	3.6557	3.9722
3.1383	5.2965	6.2894	1.7673	12.4405	7.0692	24.8809	15.9057	37.3214	28.2769	49.7618	2.483	5.556

Tabla 3.1: Muestra del dataset de viga con esfuerzos constantes

3.3. Viga simplemente apoyada

En el segundo de los problemas se usará también una viga de 20 cm de radio y material arbitrario, esta vez apoyada en sus dos extremos con un apoyo fijo y uno móvil. La carga se va aplicar es un punto intermedio de la viga elegido arbitrariamente, con componentes en la dirección axial, lo que generará esfuerzo normal, y en dirección vertical, que generará esfuerzo cortante y flector:

1. Longitud, $L : 4 - 20m$
2. Punto de aplicación de la carga, $x_L < L$
3. Carga axial, $P_x : 0 - 2000N$
4. Carga vertical en dirección Y negativa, $P_y : 0 - 2000N$
5. Modulo de Young, $E : 60 - 210GPa$
6. Modulo de elasticidad transversal, $G : 22 - 84GPa$



Se consideran salidas a las reacciones y deformaciones en los apoyos. El cálculo de las reacciones es directo, estableciendo equilibrio estático se obtienen expresiones sencillas como en el caso anterior. En el caso de la deformación angular en los extremos, su cálculo no es tan sencillo, debiéndose integrar la curvatura $v'' = \frac{M_f(x)}{EI_z}$ y aplicar condiciones de contorno. La expresión resultante es compleja, sirviendo como excelente punto de comparación entre el uso de polinomios y redes neuronales cuando no se conocen las expresiones matemáticas que relacionan salidas con entradas. Al igual que en el problema anterior se incluye el script de Matlab usado para la generación del dataset, y se incluye en la tabla 3.2 una fracción de éste.

Entradas					
$L(m)$	$x_L(m)$	$P_x(kN)$	$P_y(kN)$	$E(Gpa)$	$G(Gpa)$
13.2507	4.7678	0.8139	1.8337	101.1417	38.0539
8.1005	2.3787	0.5663	1.0936	108.4377	42.4223
4.5978	1.5935	1.497	1.3019	209.865	78.4405
9.0784	3.583	0.5092	0.0465	137.1112	54.6156
4.9156	1.5407	1.266	1.3554	181.0992	68.2893
12.1357	8.2871	1.9173	0.0086	62.7047	24.9678
Salidas					
$R_{Ax}(KN)$	$R_{Ay}(kN)$	$R_{By}(kN)$	$\theta_A(kN) \cdot 10^7$	$U_{Bx}(nm)$	$\theta_B \cdot 10^7$
-813.9	1173.9	659.8	-1595.1	305.3	1322.4
-566.3	772.5	321.1	-310.6	98.8	235.5
-1497	850.7	451.2	-65.1	90.4	53
-509.2	28.2	18.4	-14.2	105.9	12.4
-1266	930.6	424.8	-87.1	85.7	67.8
-1917.3	2.7	5.9	-7.7	2016.4	9.8

Tabla 3.2: Muestra del dataset de viga biapoyada

Capítulo 4

Aproximación polinómica

4.1. Aproximación de funciones

En esta sección se presenta un método para aproximación de funciones basado en polinomios. La idea es sencilla y bien conocida. El problema en concreto consiste en aproximar una función, en principio se considera escalar, $f(\boldsymbol{x})$ de variables reales, $\boldsymbol{x} \in \mathbb{R}^n$, a partir de un conjunto de datos, $\{f_i, \boldsymbol{x}_i\}_{i=1}^N$
Si no existen más restricciones, se puede tomar la aproximación de la función, $\widehat{f}$, perteneciente a cierto espacio funcional conocido, $\widehat{f} \in \mathcal{F}$. A partir de una base de este espacio funcional, se conoce entonces la forma de $\widehat{f}$ y su representación final se puede encontrar a partir de un problema de minimización de cierta distancia, que en general será la euclídea.
Formalmente, el problema se puede plantear de la siguiente manera. Sea $\mathcal{F}$ un espacio funcional con base $\{\Phi(\boldsymbol{x})\}_{i=1}^d$. Sea entonces $\widehat{f} \in \mathcal{F}$, por lo que se puede escribir $\widehat{f}$ como una combinación lineal de las funciones base, es decir

$$\widehat{f}(\boldsymbol{x}) = \sum_{i=1}^{d} \widehat{\alpha}_i \cdot \Phi(\boldsymbol{x}) \tag{4.1.1}$$

Con esta aproximación, se puede definir la siguiente función de coste

$$\eta\left(\{\widehat{\alpha}_i\}_{i=1}^d\right) = \sum_{j=1}^{N} \|f_i - \widehat{f}(\boldsymbol{x}_i)\| \tag{4.1.2}$$

donde $\|\cdot\|$ es cierta norma que represente el error que se quiere medir, que precisamente si se usa la norma dos del espacio euclidiano, la expresión que queda es precisamente la función de coste error cuadrático medio que se venía usando en las redes neuronales.

Obsérvese que la función anterior, η, depende únicamente de las $\{\alpha_i\}_{i=1}^d$, ya que $\widehat{f}$ se evalúa en cada uno de los puntos del conjunto de datos. La ecuación 4.1.2 representa la suma de los errores (medidos en la norma $\|\cdot\|$) en cada punto del conjunto de datos. A falta de más restricciones, una buena aproximación de f es precisamente minimizar ese error. Es decir, resolver el siguiente problema de optimización

$$\{\widehat{\alpha}\} = \min_{\{\alpha_i\}_{i=1}^d} \eta \tag{4.1.3}$$

El problema 4.1.3 es un problema de mínimos cuadrados cuando la norma usada es la norma euclídea, y, por tanto, tiene solución en ese caso. Así pues, la obtención de una aproximación $\widehat{f}$ de f está garantizada. Sin embargo, el problema surge ahora con el espacio funcional donde buscamos esta aproximación, $\mathcal{F}$.
La elección del espacio funcional puede deberse tanto a razones físicas en el caso de aproximación de datos reales, como a razones numéricas. Por ejemplo

- Para la aproximación de cargas externas, el uso de un espacio funcional de funciones gaussianas en torno a cierto a conjunto de centros parece natural, al poder considerar las gaussianas como cargas en el entorno de su centro (lo grande que se considere este entorno depende de la desviación típica) (ver figura 4.1.1).
- Es bien conocido, como se verá a continuación, el uso generalizado de polinomios para aproximacón por mínimos cuadrados. Sin embargo, polinomios de alto orden pueden dar lugar a aproximaciones demasiado inestables como consecuencia de los errores inevitables en las mediciones (ver figura 4.1.2).

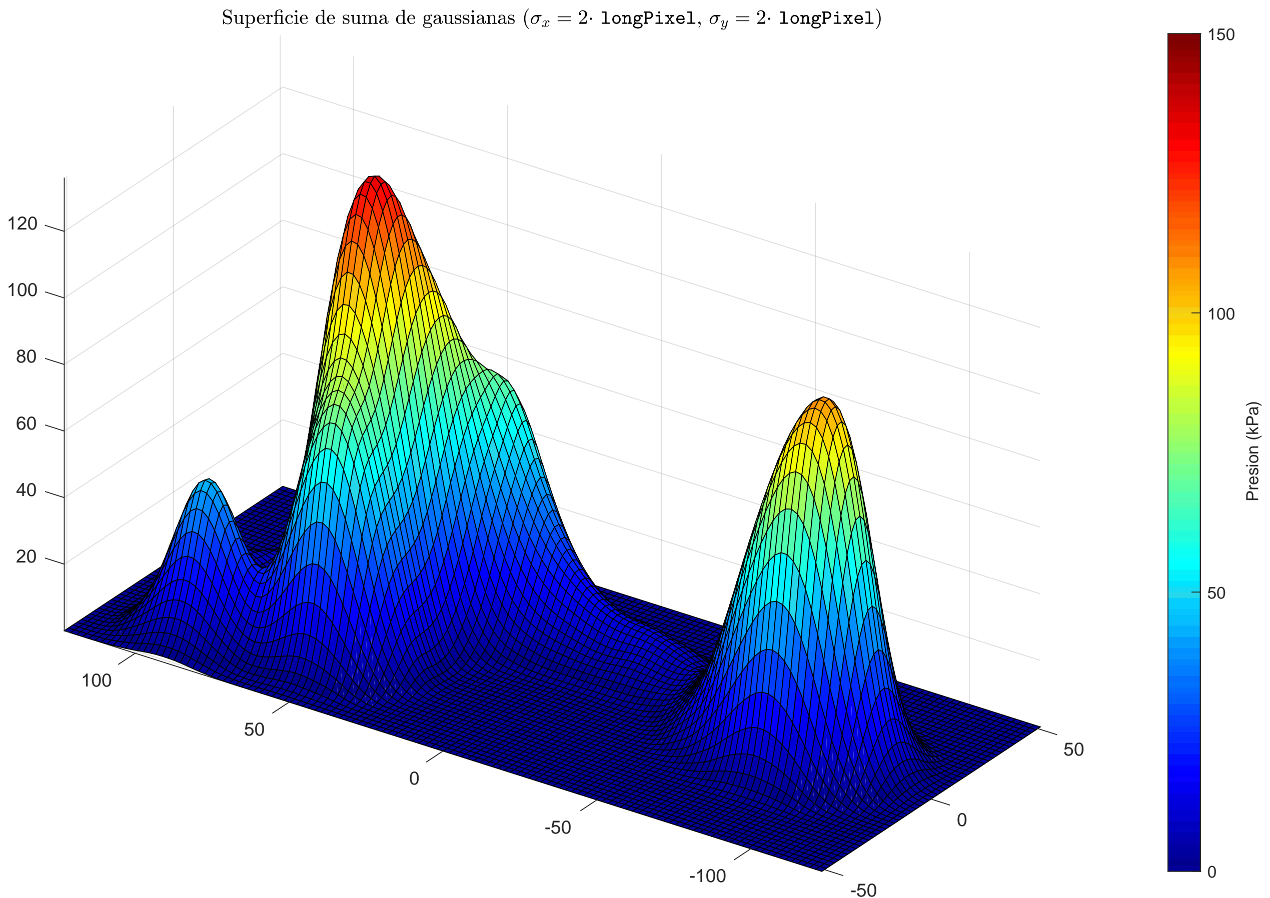


Figura 4.1.1: Modelización del campo de presiones generado durante la pisada mediante funciones gaussianas.

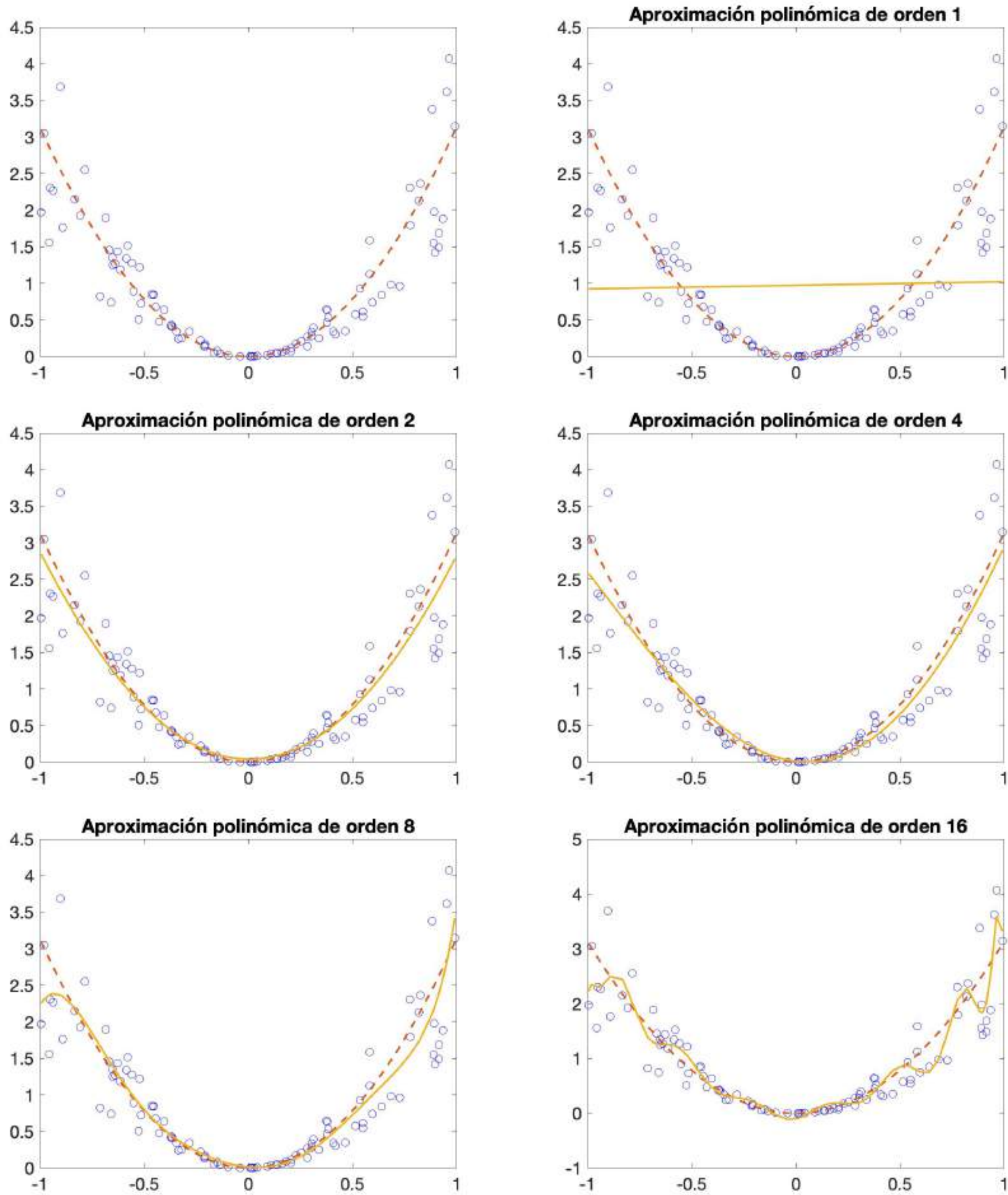


Figura 4.1.2: Aproximación polinómica de la curva $f(x) = \pi x^2$ con un conjunto de 100 datos con un error aleatorio máximo del 50 %. Los críulos azules representan los datos ; la línea discontinua naranja, la función real, $f(x)$; finalmente, en amarillo continuo se representan las distintas aproximaciones

4.1.1. Aproximación polinómica

Para el problema de aproximación planteado en 4.1.3 se tiene una aproximacón polinómica si se utiliza un espacio funcional de polinomios de grado n_d para variables de dimensión n, $\mathcal{P}_{n_d}^n$. Es decir, el conjunto de funciones base $\{\Phi(\boldsymbol{x})\}_{i=1}^d$ debe tomarse de entre las posibles bases de polinomios de orden n_d con variables de dimensión n. En el caso particular de funciones

en $\mathbb{R}$, la dimensión de la base coincide con el orden del espacio de polinomios más uno, es decir, $d = n_d + 1$. Por ejemplo,

$$\{\varPhi(x)\}_{i=1}^{d} = \{\varPhi(x)\}_{i=1}^{n_d+1} = \left\{1, x, x^2, x^3, ..., x^{n_d}\right\} \tag{4.1.4}$$

En el caso de funciones de variables en $\mathbb{R}^n$, la base sigue siendo sencilla de calcular, pudiéndose escribir de la forma

$$\begin{aligned}\{\varPhi\}_{i=1}^{d} = \{\, & 1, x_1, x_2, ..., x_n, x_1^2, x_1x_2, ..., x_1x_n, x_2^2, x_2x_3, ..., x_2x_n, ..., x_n^2, \\ & x_1^3, x_1^2x_2, x_1^2x_3, ..., x_1^2x_n, x_1x_2^2, x_1x_3^2, ..., x_n^3, ..., x_1^{n_d}, x_2^{n_d}, ..., x_n^{n_d} \,\}\end{aligned} \tag{4.1.5}$$

Obsérvese que la dimensión de la base no es lineal con el orden de los polinomios del espacio funcional. Un cálculo combinatorio básico da la relación de la dimensión de la base, d, el orden de los polinomios, n_d, y la dimensión de las variables independientes, n,

$$d = 1 + \sum_{i=1}^{n_d} \left(\!\binom{n}{i}\!\right) = 1 + \sum_{i=1}^{n_d} \binom{i+n-1}{i} = 1 + \sum_{i=1}^{n_d} \frac{(i+n-1)!}{i! \cdot (n-1)!} \tag{4.1.6}$$

donde $\left(\!\binom{n}{i}\!\right)$ es el número de combinaciones con repetición de i elementos escogidos de un conjunto con n elementos, es decir, el número de funciones base de grado i con variables de dimensión n.
La ecuación 4.1.6 es importante tenerla en cuenta de cara a procesos computacionales, pues tanto el tamaño (d), como el orden de las funciones base a la hora de almacenarlas es relevante para optimizar el proceso, tanto en tiempo como en memoria.
Teniendo en cuenta entonces la base para variables de dimensión n, 4.1.5, cualquier función perteneciente al espacio funcional $\mathcal{P}_{n_d}^{n}$ se puede escribir como

$$\widehat{f}(\boldsymbol{x}) = \sum_{i=1}^{d} \widehat{\alpha}_i \varPhi_i(\boldsymbol{x}) \quad \forall \widehat{f} \in \mathcal{P}_{n_d}^{n} \tag{4.1.7}$$

En la ecuación 4.1.8 se muestra la expresión genérica que tomaría una salida cualquiera del modelo, $\widehat{y}_i$, para n entradas y grado gr. Las variables α_j son los parámetros del modelo, e irán cambiando de valor durante la minimización para adaptarse a las observaciones dadas.

$$\widehat{y}_i = \sum \alpha_j \cdot \prod_{k=1}^{n} x_k^{q_k}, \quad \forall \sum q_k \leq gr \tag{4.1.8}$$

4.1.2. Teorema de Stone-Weirstrass

Como se ha descrito anteriormente, la elección del espacio funcional es determinante a la hora de obtener una buena aproximación para un conjunto de datos dado. Por lo que las preguntas inmediatas tras elegir los polinomios es cómo de buena es esta elección y para qué casos sirve. Las respuestas a estas preguntas es precisamente el teorema de Stone-Weirstrass.
El teorema de Stone-Weirstrass establece que el conjunto de funciones polinómicas es denso en el conjunto de funciones continuas. Es decir, que cualquier función continua puede ser aproximada por un polinomio tanto como se quiera. Por tanto, si se asume que la función que

se quiere aproximar es continua, la elección de un espacio de polinomios de grado suficiente para el espacio funcional del algoritmo dado por 4.1.3 dará un resultado tan bueno como se quiera.
Es importante mencionar que "tan bueno como se quiera" es un resultado teórico, ya que cualquier medición real lleva consigo un error, por lo que el método de aproximacón explicado no es exacto y, como consecuencia, puede haber problemas como el mencionado de las inestabilidades numéricas (ver figura 4.1.2). Además, en la práctica el grado del polinomio no se puede hacer tan grande como se quiera, sino que estará limitado por razones computacionales (ya sea de tiempo o memoria).
A pesar de los problemas planteados, queda claro que la aproximación polinómica es lo que se entiende como aproximación universal, al igual que lo son otros conjuntos de funciones, como las redes neuronales *feedforward*, para los que existen teoremas análogos al teorema de Stone-Weirstrass.

Para el entrenamiento y obtención de resultados se utilizan los códigos 3 y 4 del apéndice A. Por un lado, se tiene la función de entrenamiento 'ml_polynomial.m', que construye la función de coste como 'function handle' de los parámetros α, los inicializa, y llama a la función fmincon de Matlab para realizar la minimización. En el siguiente script, 'train_routine.m', se abren los datasets deseados y se eligen el número de observaciones y el grado de los polinomios, para así llamar a la función previa y realizar el entrenamiento, posteriormente se sacan los errores de los datos en el conjunto de test.

4.2. Viga con esfuerzos constantes

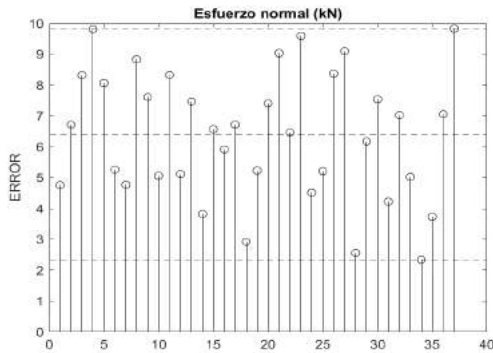
En el capítulo 5 se definen con precisión las relaciones matemáticas entres las entradas y las salidas, en dichas expresiones, se aprecia como los términos relacionados con el material, E y G, aparecen en el denominador, por tanto, sabiendo la expresiones que forman los polinomios, se sabe que introduciendo en el dataset, los inversos de estas variables, en vez de el valor directamente, se puede llegar con facilidad a la solución exacta. Por ello, se han generado dos datasets de entrada, uno con E y G, y otro modificado con 1/E y 1/G. Si la minimización se realiza con el mismo grado de polinomios, el mismo tamaño de dataset y los mismos parámetros de minimización, las diferencias son muy notables. Sobre lo de usar como entrada la inversa de cualquier variable, es perfectamente valido, y en muchas ocasiones, apropiado incluso, siempre y cuando la variable en cuestión no pueda tomar el valor de cero, como es el caso de E y G, que su nulidad no tiene sentido físico.

- Tamaño del dataset: 247
- Grado: 4
- Tolerancia de paso: 1e-21
- Máximas evaluaciones de la función: 400000

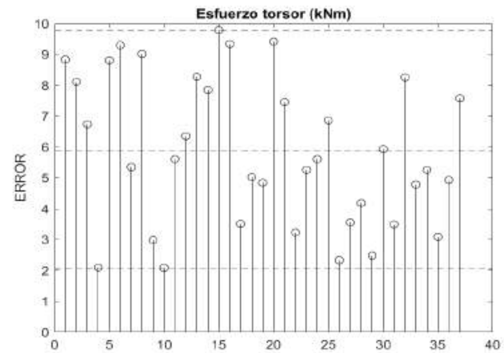
4.2.1. Sin modificar

A partir de aquí, y para mostrar la bondad de los resultados, se utilizarán gráficos del mismo tipo, que muestran por cada observación del conjunto de test la diferencia entre su valor teórico y el valor calculado por el modelo. Cada observación corresponde con un círculo del que sale una linea perpendicular al eje de abscisas y que llega hasta él. Cuanto mayor sea la distancia entre el círculo de la observación y el eje de abscisas, peor se comportará el modelo para dicha observación. Además, se muestran tres lineas discontinuas perpendiculares al eje de ordenadas, indicando el error mínimo, máximo y medio.

Las figuras desde la 4.2.1a hasta 4.2.7a muestran los gráficos explicados anteriormente para el modelo polinómico de la viga empotrada con el dataset sin modificar. Los tres primeros gráficos corresponden con los errores en el cálculo de los esfuerzos, normal, torsor y flector. Las gráficas restantes corresponden con los desplazamientos, longitudinales y angulares, en distintos puntos de la viga: $0.25L,\ 0.5L,\ 0.75L\ y\ L$. E las gráficas se puede ve claramente como los resultados no son buenos, y serán claramente mejorados con el dataset modificado.

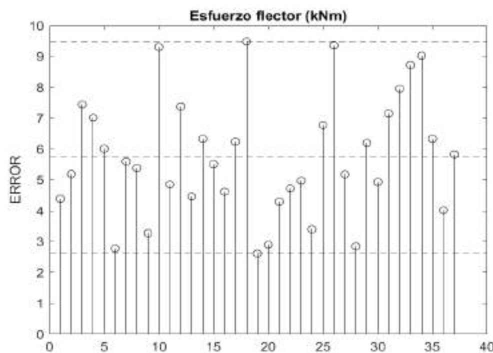


(a) Error (Esfuerzo normal)

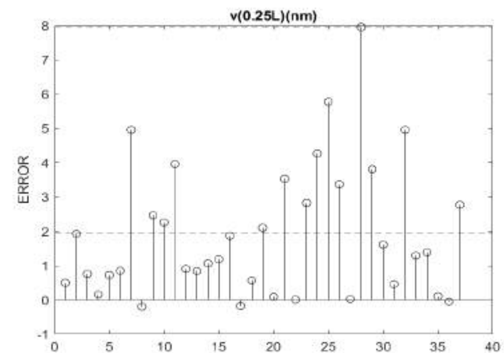


(b) Error (Esfuerzo torsor)

Figura 4.2.1: Gráficas de resultados viga empotrada

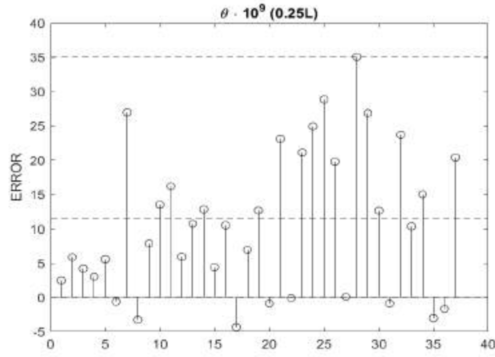


(a) Error (Esfuerzo flector)

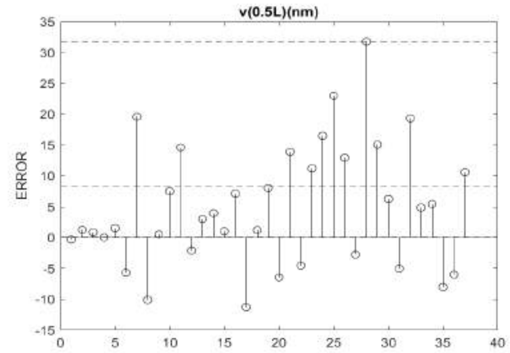


(b) Error (Desplazamiento vertical)

Figura 4.2.2: Gráficas de resultados viga empotrada

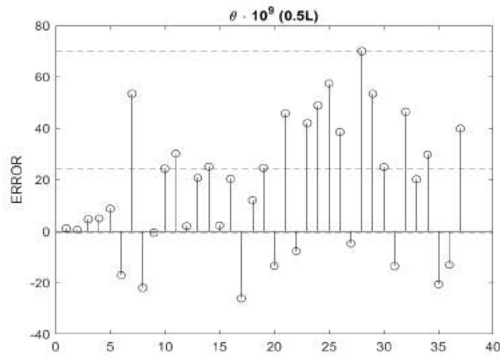


(a) Error (Ángulo de flexión)

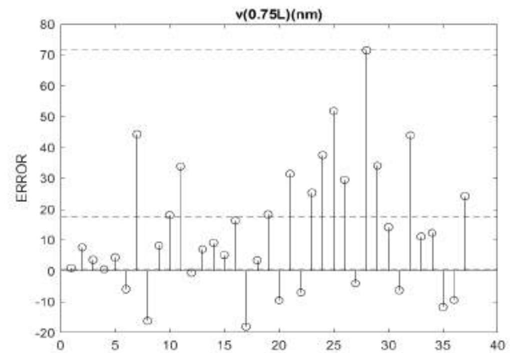


(b) Error (Desplazamiento vertical)

Figura 4.2.3: Gráficas de resultados viga empotrada

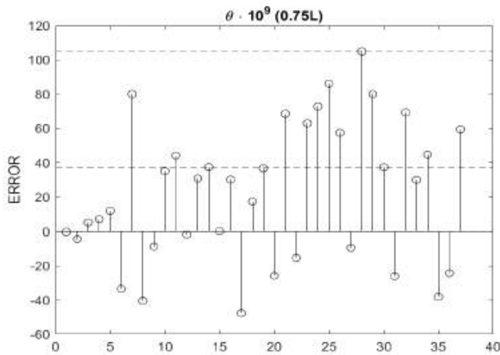


(a) Error (Ángulo de flexión)

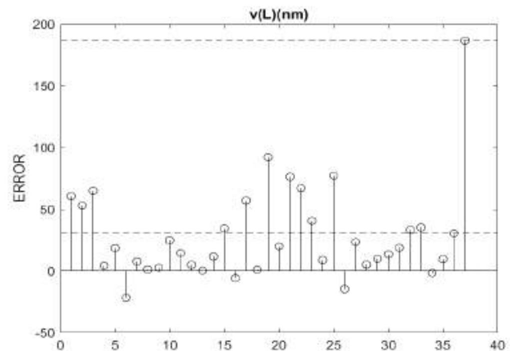


(b) Error (Desplazamiento vertical)

Figura 4.2.4: Gráficas de resultados viga empotrada

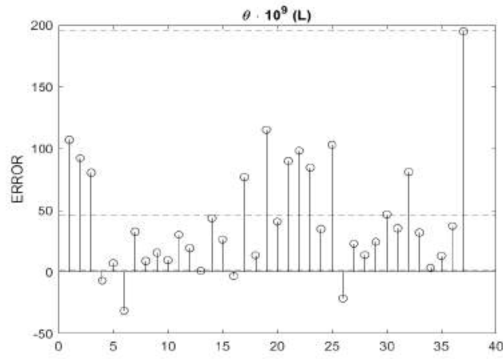


(a) Error (Ángulo de flexión)

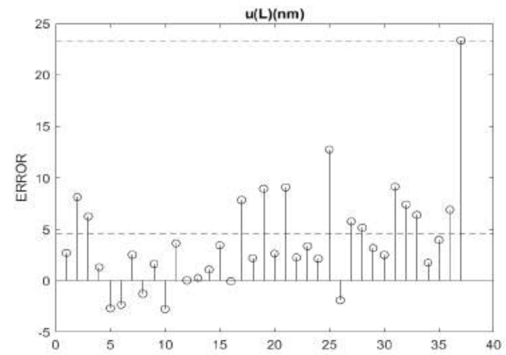


(b) Error (Desplazamiento vertical)

Figura 4.2.5: Gráficas de resultados viga empotrada

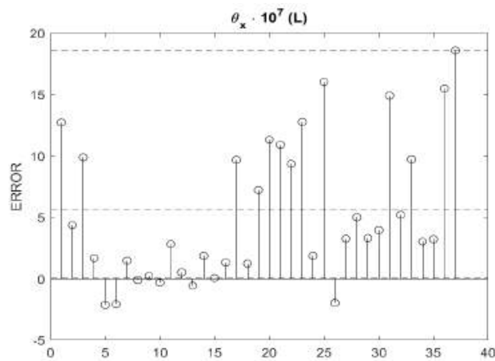


(a) Error (Ángulo de flexión)



(b) Error (Desplazamiento horizontal)

Figura 4.2.6: Gráficas de resultados viga empotrada



(a) Error (Ángulo de torsión)

Figura 4.2.7: Gráficas de resultados viga empotrada

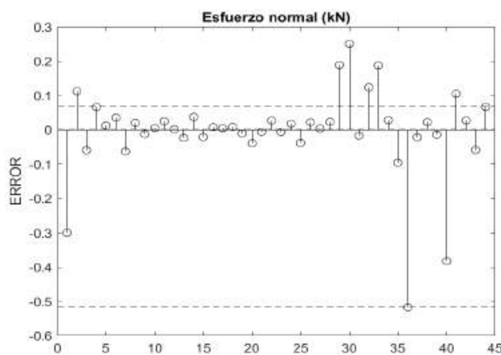
Para cada modelo se mostrará además una tabla como la 4.1, mostrando los valores numéricos reflejados por lineas discontinuas en las gráficas mostradas previamente.

	Error absoluto (Test)		
Salidas	Máximo	Mínimo	Media
$N(kN)$	9.8252	2.3405	6.3822
$M_t(kNm)$	9.7884	2.0852	5.8739
$M_f(kNm)$	9.4831	2.6126	5.7338
$v(0.25L)(nm)$	7.9552	0.0087	1.9371
$\theta \cdot 10^9(0.25L)$	35.08	0.0896	11.5105
$v(0.5L)(nm)$	31.7405	0.0585	8.1865
$\theta \cdot 10^9(0.5L)$	70.0059	-0.5277	24.0802
$v(0.75L)(nm)$	71.4892	0.548	17.7252
$\theta \cdot 10^9(0.75L)$	104.9318	0.1482	37.4419
$v(L)(nm)$	186.5034	0.3386	31.1455
$\theta \cdot 10^9(L)$	194.9236	0.9429	45.8471
$u(L)(nm)$	23.3515	0.0388	4.5561
$\theta_x \cdot 10^7(L)$	18.5963	0.065	5.6777

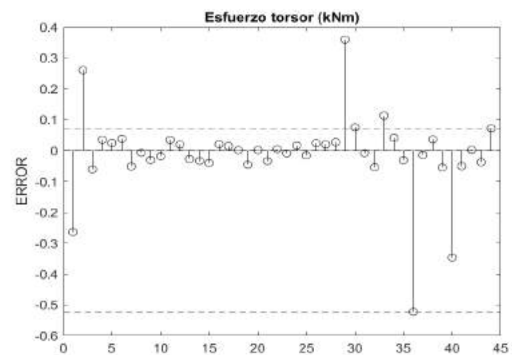
Tabla 4.1: Errores absolutos en las predicciones de la red para el problema de la viga con esfuerzos constantes usando aproximación polinómica en el dataset sin modificar

4.2.2. Modificado

Las gráficas que se muestran a continuación son equivalentes a las del modelo anterior, excepto que en este caso, el dataset de variables de entrada ha sido modificado para facilitar la labor del modelo, por tanto, los errores son considerablemente menores, por ejemplo, en el giro de flexión en el extremo de la barra, el error medio es unas 25 veces menor en este caso. Además, se observan unas distribuciones bastante más amontonadas en el eje de abscisas, donde solo son unos pocos valores los que se aproximan incorrectamente.

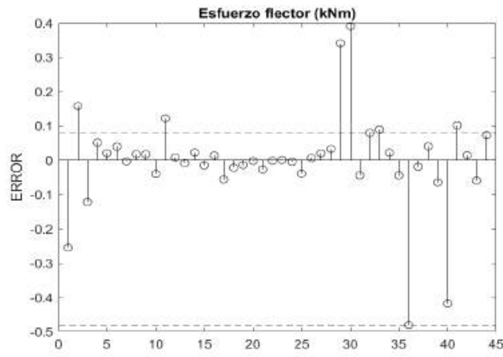


(a) Error (Esfuerzo normal)

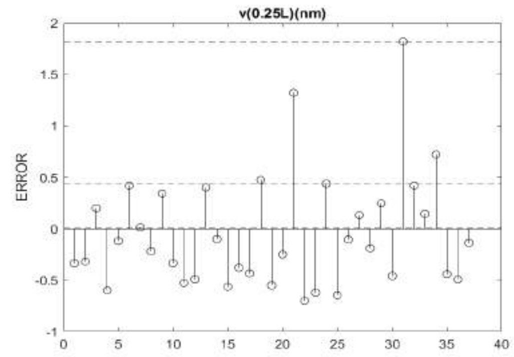


(b) Error (Esfuerzo torsor)

Figura 4.2.8: Gráficas de resultados viga empotrada

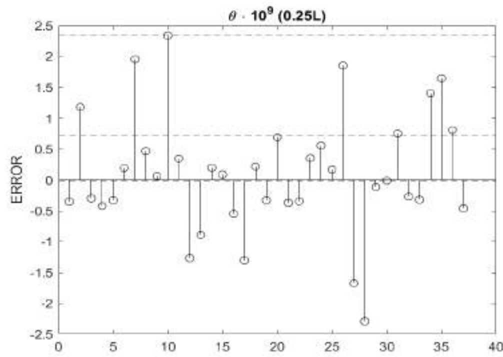


(a) Error (Esfuerzo flector)

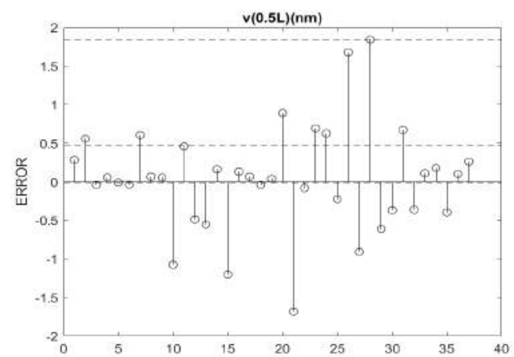


(b) Error (Desplazamiento vertical)

Figura 4.2.9: Gráficas de resultados viga empotrada

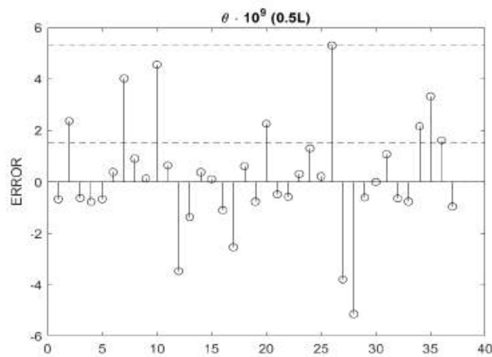


(a) Error (Ángulo de flexión)

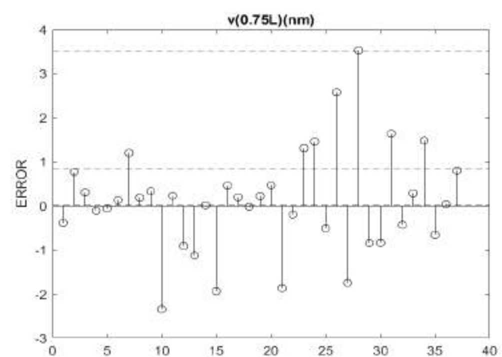


(b) Error (Desplazamiento vertical)

Figura 4.2.10: Gráficas de resultados viga empotrada

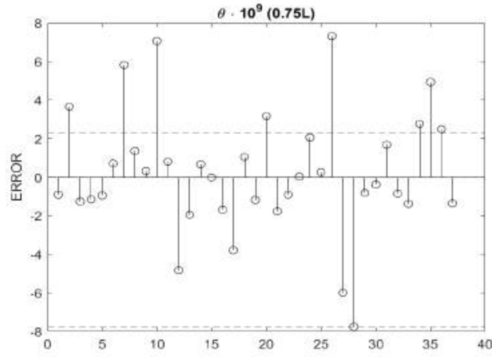


(a) Error (Ángulo de flexión)

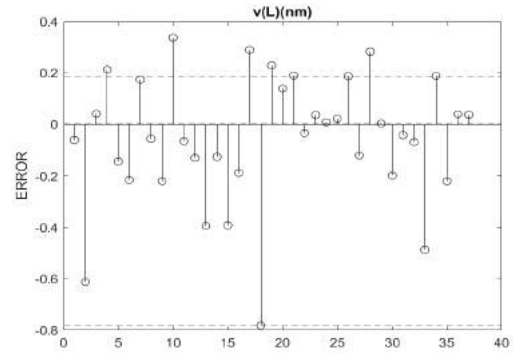


(b) Error (Desplazamiento vertical)

Figura 4.2.11: Gráficas de resultados viga empotrada

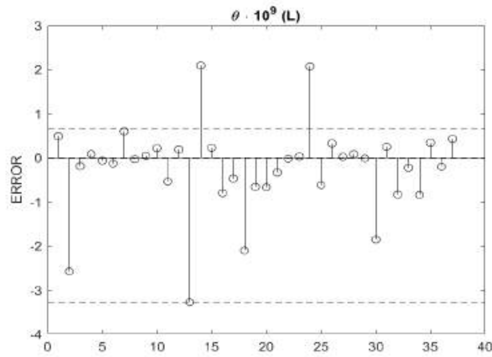


(a) Error (Ángulo de flexión)

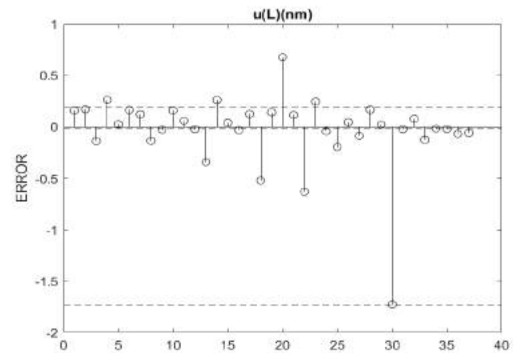


(b) Error (Desplazamiento vertical)

Figura 4.2.12: Gráficas de resultados viga empotrada

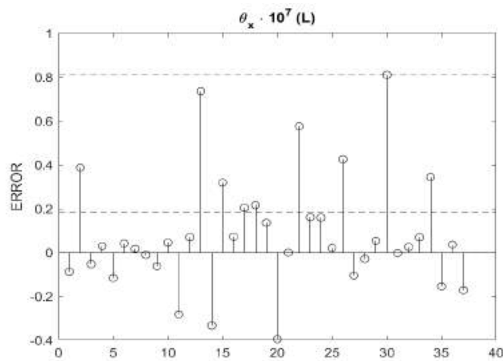


(a) Error (Ángulo de flexión)



(b) Error (Desplazamiento horizontal)

Figura 4.2.13: Gráficas de resultados viga empotrada



(a) Error (Ángulo de torsión)

Figura 4.2.14: Gráficas de resultados viga empotrada

	Error absoluto (Test)		
Salidas	Máximo	Mínimo	Media
$N(kN)$	-0.5172	0.0018	0.0708
$M_t(kNm)$	-0.5226	0.0016	0.068
$M_f(kNm)$	-0.4805	0.0007	0.0777
$v(0.25L)(nm)$	1.8196	0.0132	0.4346
$\theta \cdot 10^9(0.25L)$	2.3377	-0.012	0.7251
$v(0.5L)(nm)$	1.8436	-0.0086	0.4754
$\theta \cdot 10^9(0.5L)$	5.301	-0.0016	1.5289
$v(0.75L)(nm)$	3.5277	0.0144	0.8527
$\theta \cdot 10^9(0.75L)$	-7.7405	-0.0017	2.2948
$v(L)(nm)$	3.6955	0.0081	0.6383
$\theta \cdot 10^9(L)$	-6.0501	-0.1338	1.8242
$u(L)(nm)$	-0.7721	-0.01	0.2593
$\theta_x \cdot 10^7(L)$	-1.2195	-0.0112	0.3562

Tabla 4.2: Errores absolutos en las predicciones de la red para el problema de la viga con esfuerzos constantes usando aproximación polinómica en el dataset modificado

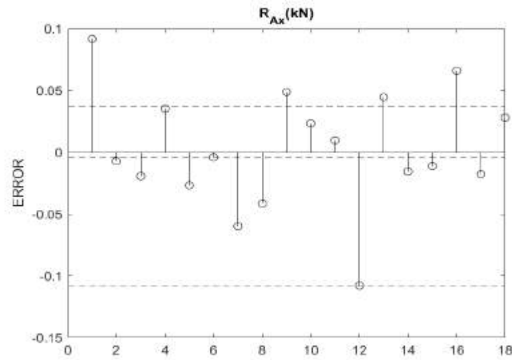
4.3. Viga simplemente apoyada

En este caso, para ahorrar un gran tiempo computacional y conseguir unos resultados aceptables, se decide prescindir de los cambios en las constante del material, manteniéndolas fijas, con E igual a 210Gpa y ν igual a 0.35. Además, para mejorar los resultados se incluye como entrada la inversa de la longitud de la viga, que aparecerá al establecerse equilibrio estático de momentos, y como dicha magnitud es forzosamente mayor que cero, no supondrá ningún conflicto. No obstante, es claro que a pesar de que con las reacciones se obtienen muy buenos resultados, las expresiones de los ángulos y desplazamientos son más complejas, muy lejos de la forma polinomial, por lo tanto, con polinomios de pequeño orden, el resultado dista de ser perfecto.

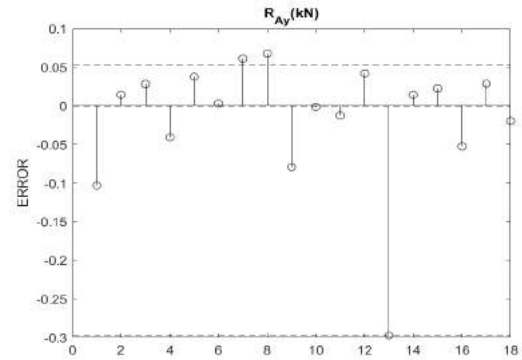
- Tamaño del dataset: 176
- Grado: 4
- Tolerancia de paso: 1e-25
- Máximas evaluaciones de la función: 600000

Los resultados para este modelo diferirán de los presentados anteriormente en que el problema es distinto, por lo que se representarán otras variables de salida. Las tres primeras corresponden con las reacciones en los apoyos, y su aproximación será buena, obteniéndose unos errores muy reducidos en todo el conjunto. La gráfica de la figura 4.3.3a muestra los errores para el desplazamiento horizontal en el apoyo simple, y su aproximación será algo

más floja que con las reacciones. Las otras dos gráficas restantes corresponden con las deformaciones angulares en los apoyos y su aproximación es mala, ya que corresponden con expresiones analíticas complejas, a las que el modelo no es capaz de llegar.

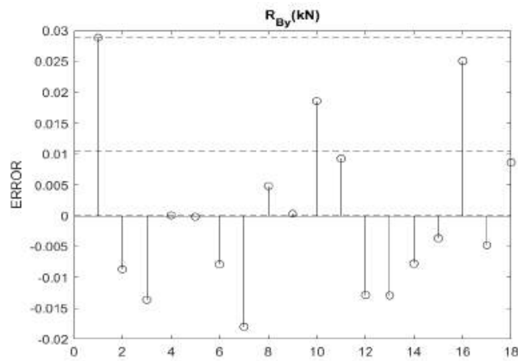


(a) Error (R_{Ax})

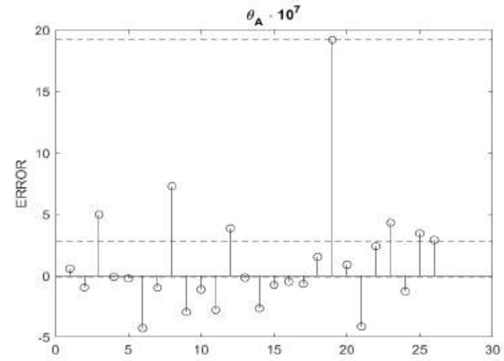


(b) Error (R_{Ay})

Figura 4.3.1: Gráficas de resultados viga simplemente apoyada

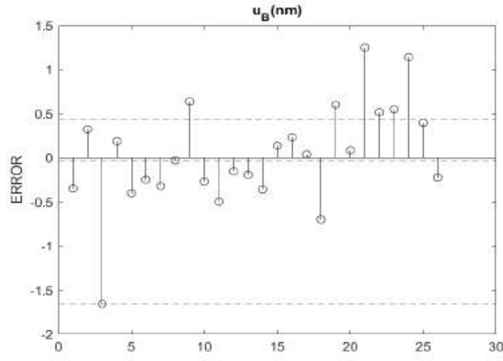


(a) Error (R_{By})

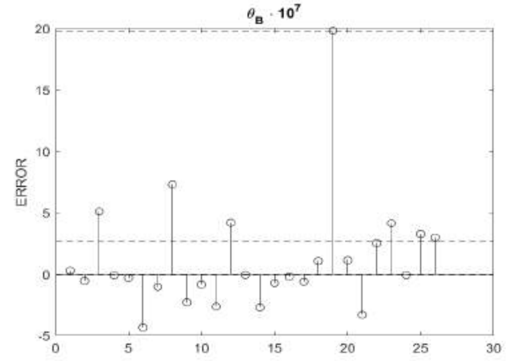


(b) Error (Ángulo en A)

Figura 4.3.2: Gráficas de resultados viga simplemente apoyada



(a) Error (Desplazamiento horizontal en B)



(b) Error (Ángulo en B)

Figura 4.3.3: Gráficas de resultados viga simplemente apoyada

	Error absoluto (Test)		
Salidas	Máximo	Mínimo	Media
$R_{Ax}(kN)$	-0.1147	0.0005	0.0595
$R_{Ay}(kN)$	0.2975	-0.0019	0.0572
$R_{By}(kN)$	-0.0865	0.0014	0.0297
$\theta_A \cdot 10^7$	19.225	-0.0818	2.8378
$u_B(nm)$	-1.6549	-0.0291	0.4357
$\theta_B \cdot 10^7$	19.8444	-0.0482	2.7129

Tabla 4.3: Errores absolutos en las predicciones de la red para el problema de la viga biapoyada usando aproximación polinómica

Capítulo 5

Descenso del gradiente vs Levenberg-Marquardt

El presente capítulo mostrará los resultados que se obtienen cuando se usan redes neuronales frente a los problemas propuestos en el capítulo 3. Además, servirá a modo de estudio comparativo de los algoritmos de minimización expuestos en la teoría: Descenso del gradiente y Levenberg-Marquardt [24].

Los códigos del 5 al 8 del apéndice A constituyen y entrenan las redes. La arquitectura de la red consta de una sola capa oculta de numero de neuronas variable con función de activación tangente hiperbólica. Una pareja de códigos se utiliza para entrenar con el descenso del gradiente, donde se pueden definir los valores de la tasa de aprendizaje, el parámetro de regularización y el momento. Por otro lado, en los scripts de minimización por Levenberg-Marquardt se podrá elegir el parámetro μ y los factores de incremento y decremento de éste.

5.1. Viga con esfuerzos constantes

5.1.1. Descenso del gradiente

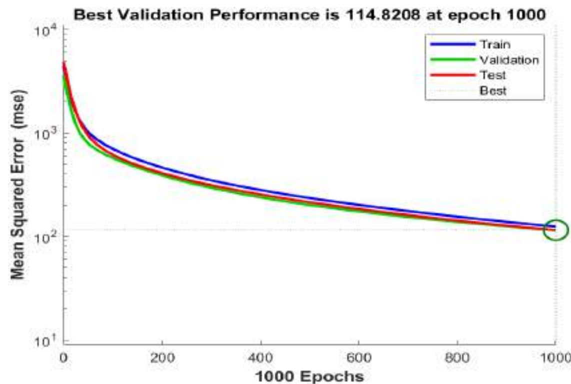
- Tamaño del dataset: 150
- Neuronas de la capa oculta: 8
- Epoch máximas: 1000
- $\alpha = 0.00001$
- $\lambda = 0.001$
- Momento, $\gamma = 0.8$

De igual forma que con los modelos polinómicos, se van a representar las gráficas con los errores en las variables de salida del conjunto de test. Además, se va añadir para cada caso

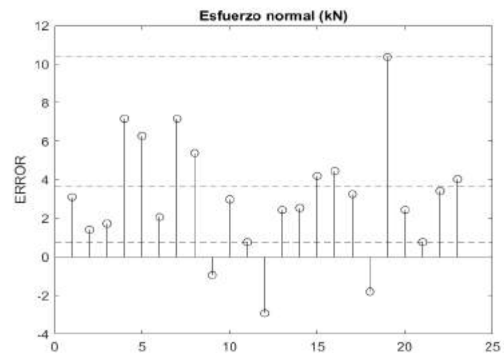
un gráfico como el de la figura 5.1.1a, que representa la evolución del error para los conjuntos de entrenamiento, validación y test durante todo el entrenamiento. En un gráfico de estas características se pueden identificar fenómenos importantes como el overfiting, si en un punto durante el entrenamiento el mse de validación se empieza a incrementar mientras que el de entrenamiento se sigue reduciendo, también se podría identificar un estancamiento en un mínimo local, si el error llega un punto en el que se mantiene plano.

Es importante mencionar que, a pesar de que en las redes en las que se está usando el descenso del gradiente se presenten arquitecturas con menos neuronas y datasets más pequeños, aumentar estas cifras y usar redes equivalentes a las que se ven con el algoritmo de Levenberg-Marquardt, no mejoraría los resultados, todo lo contrario, se observa una evolución del error que sigue una función lineal ascendente, en el que el mejor rendimiento se consigue en la primera iteración, cuando los pesos de la red están inicializados aleatoriamente.

Sobre las gráficas de errores, entre la figura 5.1.1b y 5.1.7b, se debe comentar que los resultados son realmente malos, con errores medios que rondan entres 3 y 12 entre todas las variables de salida, y que son del mismo orden de magnitud que los propios valores de salida, lo que indica que la precisión de la red no es mucho mayor que la de cualquier sistema de generación de números aleatorios entre 0 y 100.

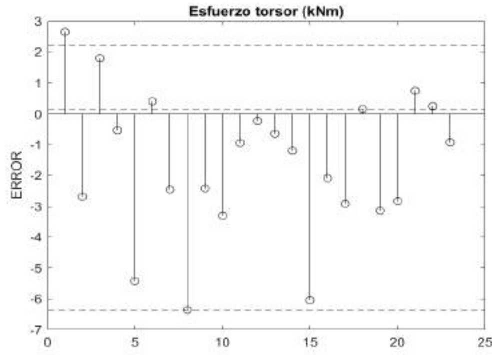


(a) Evolución del mse durante el entrenamiento

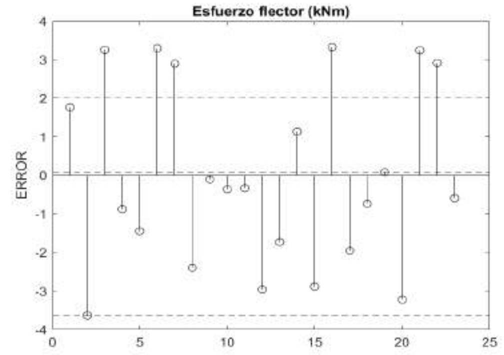


(b) Error (Esfuerzo normal)

Figura 5.1.1: Gráficas de resultados viga empotrada

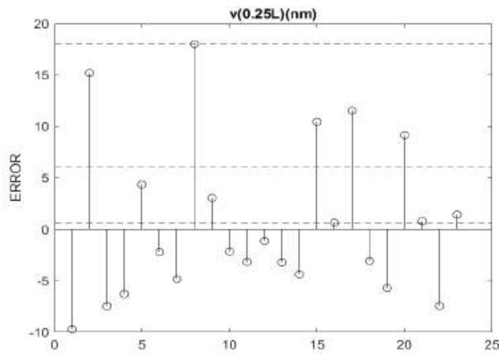


(a) Error (Esfuerzo torsor)

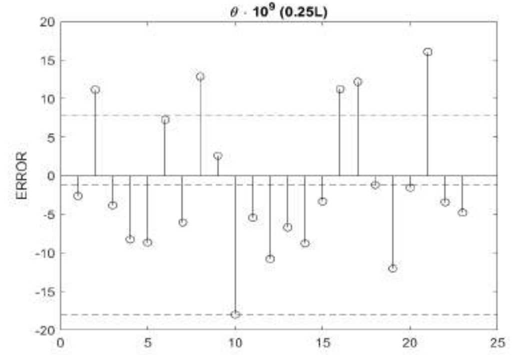


(b) Error (Esfuerzo flector)

Figura 5.1.2: Gráficas de resultados viga empotrada

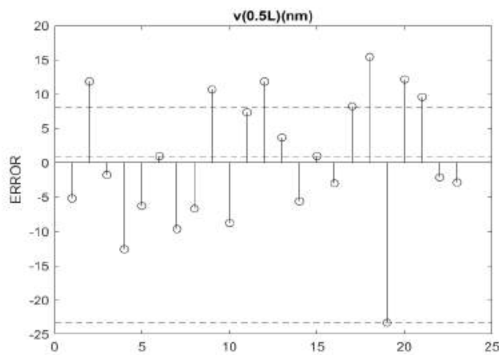


(a) Error (Desplazamiento vertical)

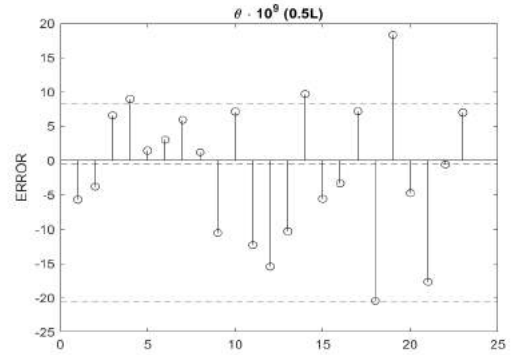


(b) Error (Ángulo de flexión)

Figura 5.1.3: Gráficas de resultados viga empotrada

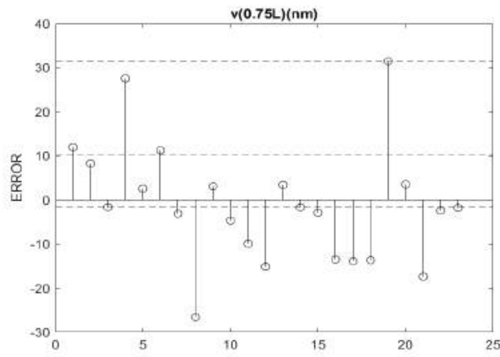


(a) Error (Desplazamiento vertical)

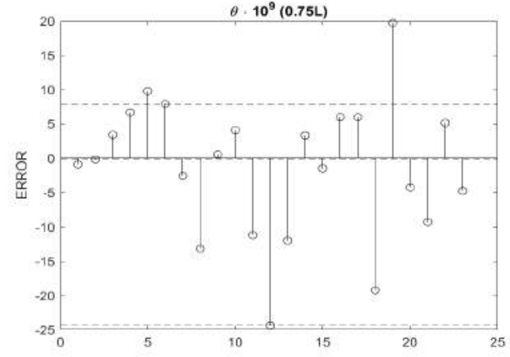


(b) Error (Ángulo de flexión)

Figura 5.1.4: Gráficas de resultados viga empotrada

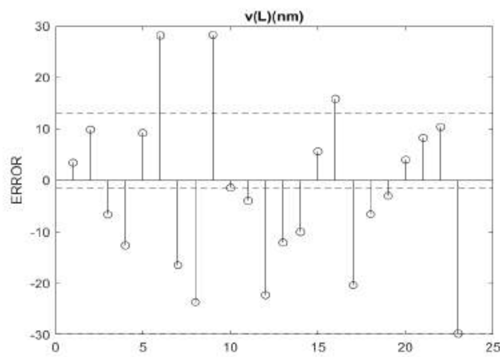


(a) Error (Desplazamiento vertical)

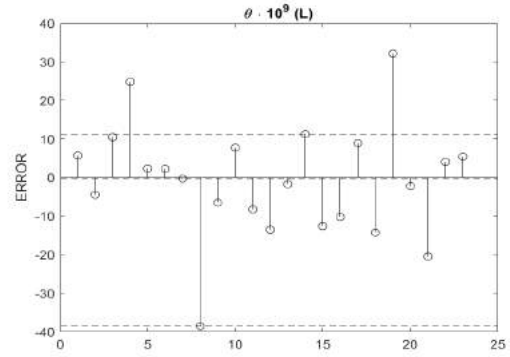


(b) Error (Ángulo de flexión)

Figura 5.1.5: Gráficas de resultados viga empotrada

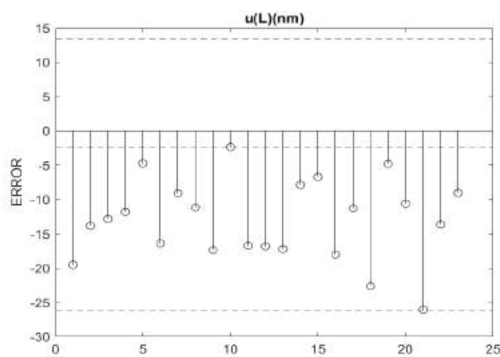


(a) Error (Desplazamiento vertical)

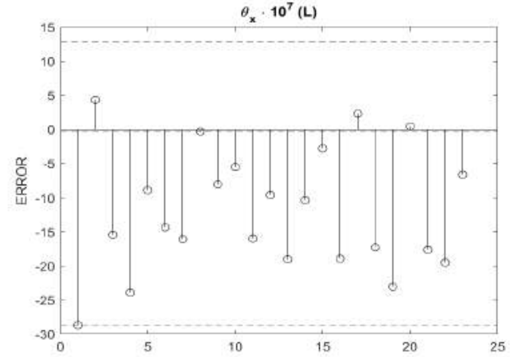


(b) Error (Ángulo de flexión)

Figura 5.1.6: Gráficas de resultados viga empotrada



(a) Error (Desplazamiento horizontal)



(b) Error (Ángulo de torsión)

Figura 5.1.7: Gráficas de resultados viga empotrada

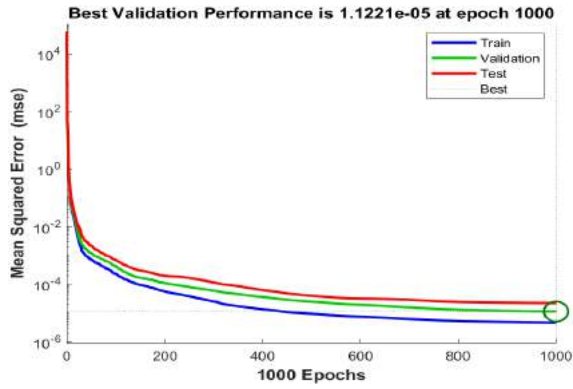
	Error absoluto (Test)		
Salidas	Máximo	Minimo	Media
$N(kN)$	10.3676	0.7745	3.6326
$M_t(kNm)$	-6.3625	0.1459	2.2293
$M_f(kNm)$	-3.631	0.0847	2.0051
$v(0.25L)(nm)$	17.9985	0.6365	6.0174
$\theta \cdot 10^9(0.25L)$	-17.9795	-1.1807	7.9382
$v(0.5L)(nm)$	-23.3172	0.9667	8.0296
$\theta \cdot 10^9(0.5L)$	-20.474	-0.5891	8.3156
$v(0.75L)(nm)$	31.4702	-1.6386	10.284
$\theta \cdot 10^9(0.75L)$	-24.3813	-0.1615	7.8254
$v(L)(nm)$	-29.7836	-1.4426	12.9845
$\theta \cdot 10^9(L)$	-38.4854	-0.2335	11.0191
$u(L)(nm)$	-26.0509	-2.3429	13.3515
$\theta_x \cdot 10^7(L)$	-28.6541	-0.3008	12.839

Tabla 5.1: Errores absolutos en las predicciones de la red para el problema de la viga con esfuerzos constantes usando el descenso del gradiente con momento

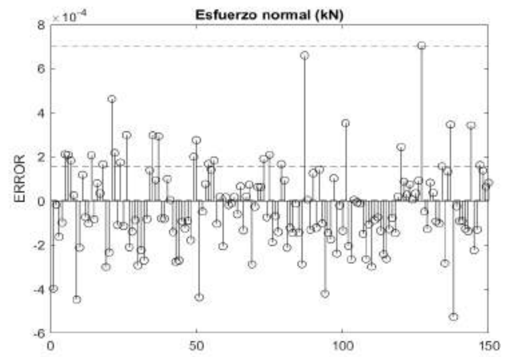
5.1.2. Levenberg-Marquardt

- Tamaño del dataset: 1000
- Neuronas de la capa oculta: 64
- Epoch máximas: 1000
- $\mu = 0.01$
- Decremento de μ: 0.8
- Incremento de μ: 10

Fijándose en los resultados, se observan grandes diferencias con respecto al descenso del gradiente. En primer lugar, en la figura 5.1.8a, se tienen unas curvas de entrenamiento con una pendiente negativa mucho más elevada, sobre todo en las 50 primeras epochs, lo que muestra un entrenamiento mucho más eficaz y resulta en una red con mejor rendimiento. Con respecto a las gráficas posteriores, se observan que los errores medios son entre 1000 y 20000 veces menores dependiendo de la variable de salida, obteniéndose unos resultados notablemente mejores, y una herramienta exitosa, de la que se podría afirmar que ha aprendido las leyes físicas asociadas al problema.

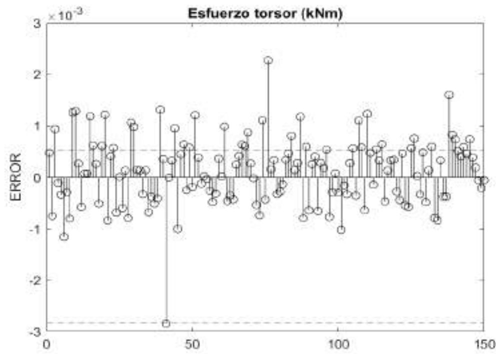


(a) Evolución del mse durante el entrenamiento

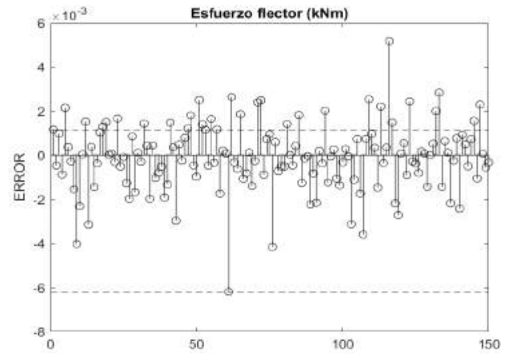


(b) Error (Esfuerzo normal)

Figura 5.1.8: Gráficas de resultados viga empotrada

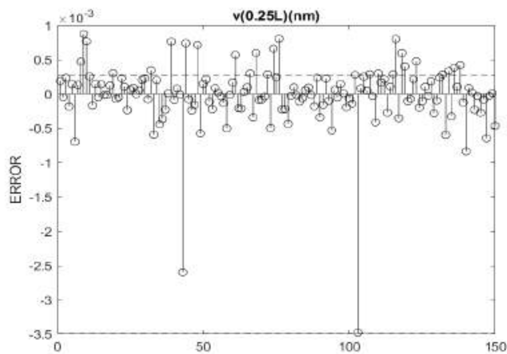


(a) Error (Esfuerzo torsor)

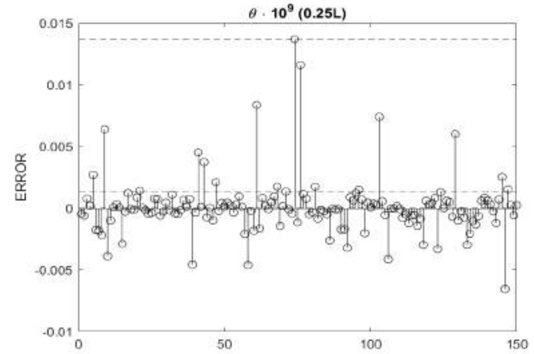


(b) Error (Esfuerzo flector)

Figura 5.1.9: Gráficas de resultados viga empotrada

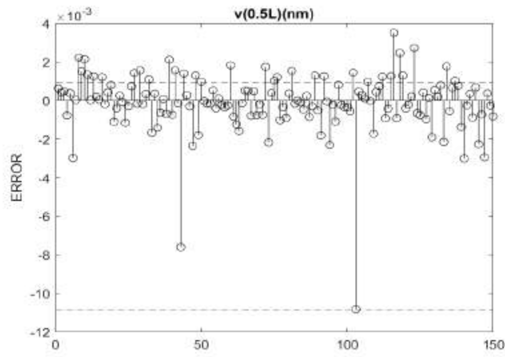


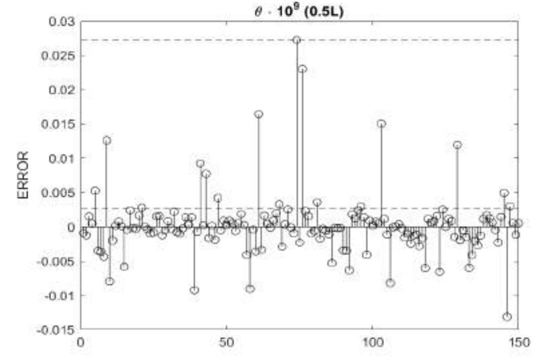
(a) Error (Desplazamiento vertical)



(b) Error (Ángulo de flexión)

Figura 5.1.10: Gráficas de resultados viga empotrada

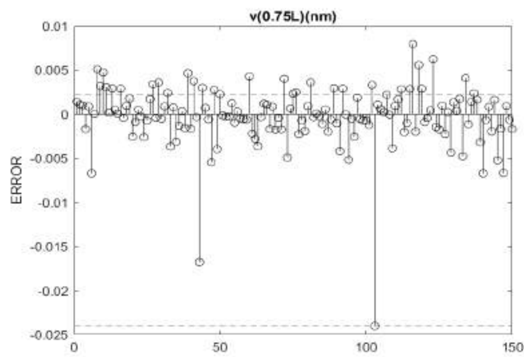


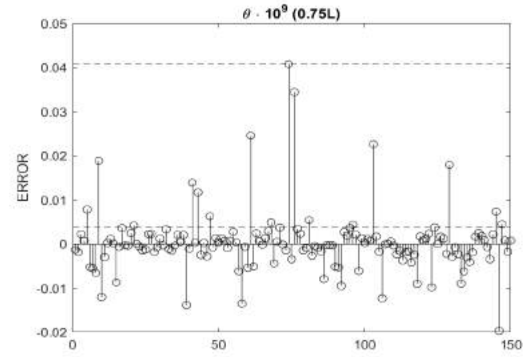


(a) Error (Desplazamiento vertical)

(b) Error (Ángulo de flexión)

Figura 5.1.11: Gráficas de resultados viga empotrada

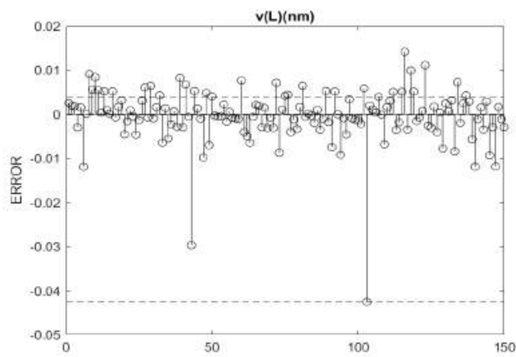


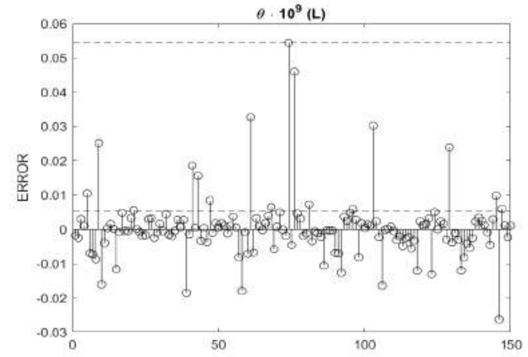


(a) Error (Desplazamiento vertical)

(b) Error (Ángulo de flexión)

Figura 5.1.12: Gráficas de resultados viga empotrada

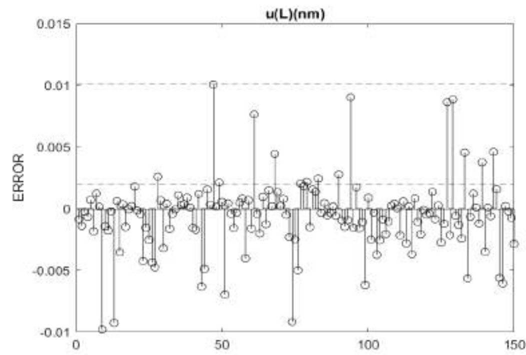




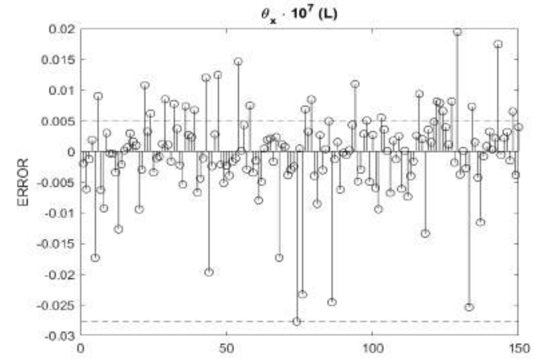
(a) Error (Desplazamiento vertical)

(b) Error (Ángulo de flexión)

Figura 5.1.13: Gráficas de resultados viga empotrada



(a) Error (Desplazamiento horizontal)



(b) Error (Ángulo de torsión)

Figura 5.1.14: Gráficas de resultados viga empotrada

	Error absoluto (Test)		
Salidas	Máximo	Mínimo	Media
$N(kN)$	0.0007	0.0000041	0.0002
$M_t(kNm)$	-0.0028	-0.0000027	0.0005
$M_f(kNm)$	-0.0062	0.0000120	0.0011
$v(0.25L)(nm)$	-0.0035	0.0000008	0.0003
$\theta \cdot 10^9(0.25L)$	0.0137	0.0000040	0.0013
$v(0.5L)(nm)$	-0.0108	0.0000135	0.001
$\theta \cdot 10^9(0.5L)$	0.0272	-0.0000171	0.0026
$v(0.75L)(nm)$	-0.0240	0.0000310	0.0022
$\theta \cdot 10^9(0.75L)$	0.0408	-0.0000328	0.0039
$v(L)(nm)$	-0.0425	-0.0000375	0.0039
$\theta \cdot 10^9(L)$	0.0544	0.0000459	0.0052
$u(L)(nm)$	0.0101	0.0000397	0.002
$\theta_x \cdot 10^7(L)$	-0.0277	0.0000679	0.005

Tabla 5.2: Errores absolutos en las predicciones de la red para el problema de la viga con esfuerzos constantes usando el algoritmo de Levenberg-Marquardt

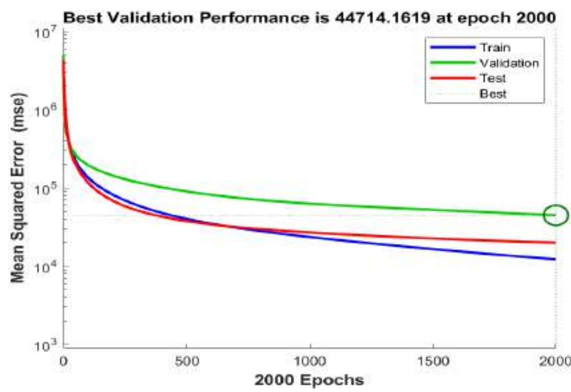
5.2. Viga simplemente apoyada

5.2.1. Descenso del gradiente

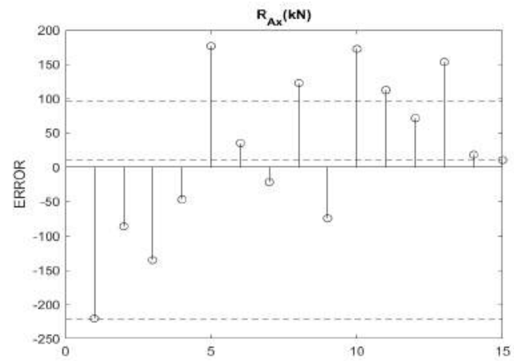
- Tamaño del dataset: 100
- Neuronas de la capa oculta: 16
- Epoch máximas: 2000
- $\alpha = 0.0000001$

- $\lambda = 0.1$
- Momento, $\gamma = 0.6$

Volviendo a hacer uso del descenso del gradiente, se vuelve a observar un entrenamiento inefectivo, con curvas muy planas. Además, se observa algo de overfiting, ya que en un momento durante el entrenamiento las curvas se separan, manteniéndose la de validación con un mse considerablemente mayor. Respecto a las gráficas del error, figuras 5.2.1b hasta 5.2.4a, presentan errores elevados y mas o menos similares, al contrario de lo que sucedía con los polinomios, en este caso y en el siguiente, no importa la complejidad de las expresiones analíticas, y las variables de salida que mejor se aproximen será tan solo una cuestión arbitraria, y no se notará una diferencia clara entre las reacciones y los desplazamientos.

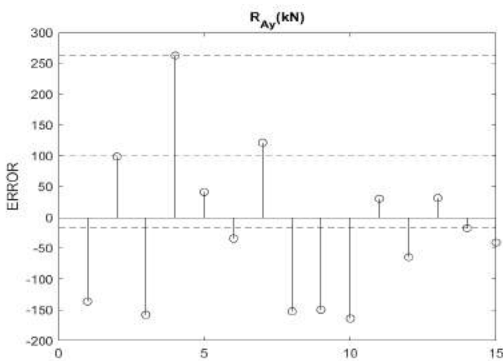


(a) Evolución del mse durante el entrenamiento

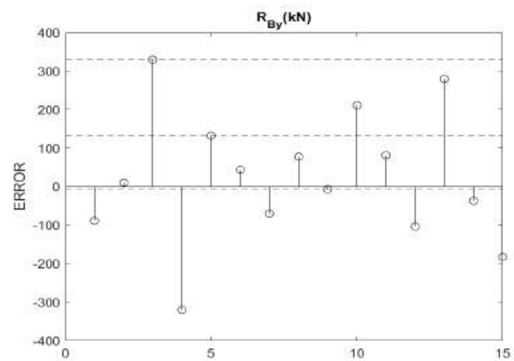


(b) Error (R_{Ax})

Figura 5.2.1: Gráficas de resultados viga simplemente apoyada

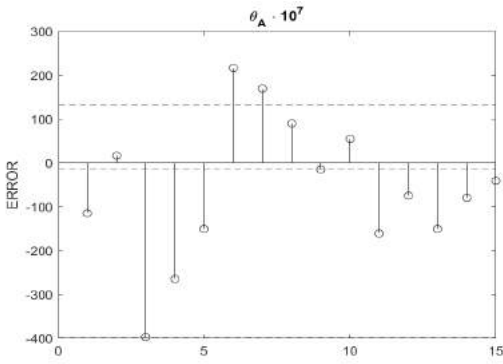


(a) Error (R_{Ay})

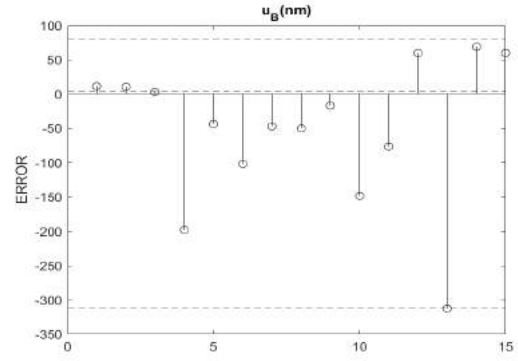


(b) Error (R_{Bx})

Figura 5.2.2: Gráficas de resultados viga simplemente apoyada

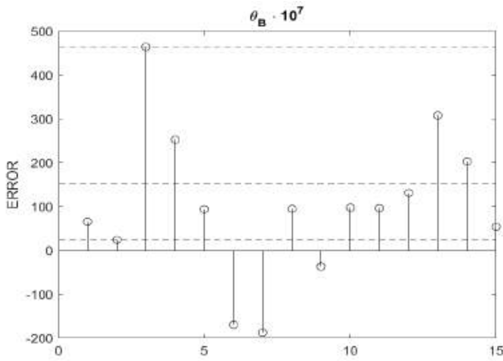


(a) Error (Ángulo en A)



(b) Error (Desplazamiento horizontal en B)

Figura 5.2.3: Gráficas de resultados viga simplemente apoyada



(a) Error (Ángulo en B)

Figura 5.2.4: Gráficas de resultados viga simplemente apoyada

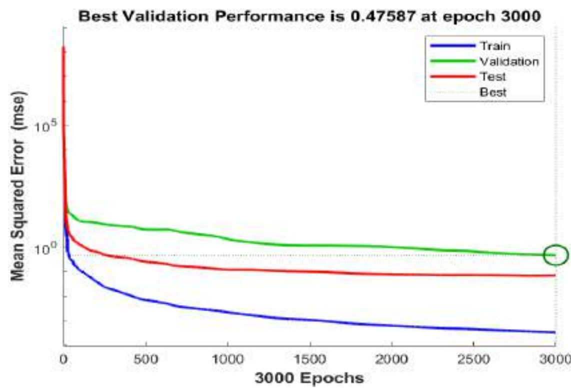
	Error absoluto (Test)		
Salidas	Máximo	Mínimo	Media
$R_{Ax}(kN)$	-220.6995	10.5392	97.2621
$R_{Ay}(kN)$	262.4967	-17.4119	100.1469
$R_{By}(kN)$	329.5801	-6.4909	131.5624
$\theta_A \cdot 10^7$	-396.9016	-14.4282	132.9392
$u_B(nm)$	-312.6731	3.3519	80.6356
$\theta_B \cdot 10^7$	464.0685	24.617	152.0756

Tabla 5.3: Errores absolutos en las predicciones de la red para el problema de la viga biapoyada usando el descenso del gradiente con momento

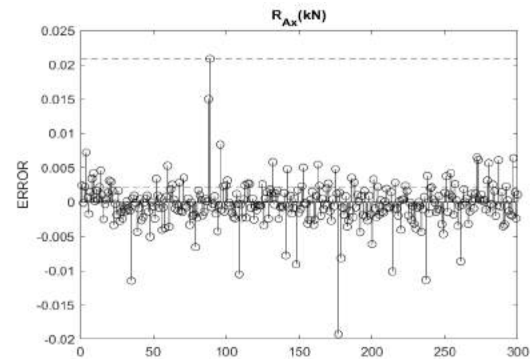
5.2.2. Levenberg-Marquardt

- Tamaño del dataset: 2000
- Neuronas de la capa oculta: 256
- Epoch máximas: 3000
- $\mu = 0.1$
- Decremento de μ: 0.3
- Incremento de μ: 3

En este caso el entrenamiento se vuelve más costoso, una arquitectura de red considerablemente más grande que en los casos anteriores dilata el tiempo de entrenamiento, y los resultados, aunque mejores que con el descenso del gradiente, no son excelentes. La figura 5.2.5a muestra también señas de overfiting. En cuanto a los gráficos de error, se tiene un error medio reducido, siendo considerablemente mayor en las figuras 5.2.7a y 5.2.8a, correspondientes con el desplazamiento angular, aunque esto no tendría porque ser así.

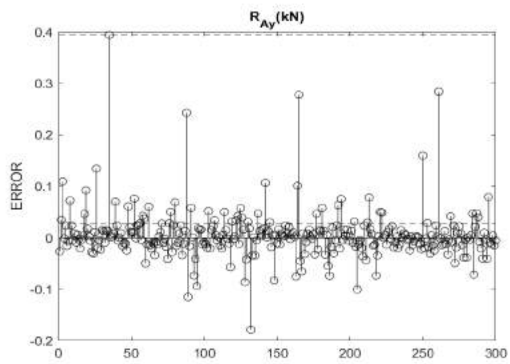


(a) Evolución del mse durante el entrenamiento

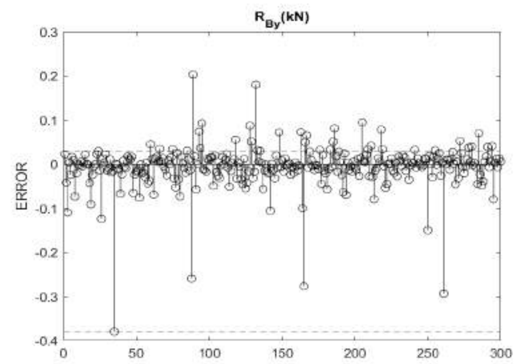


(b) Error (R_{Ax})

Figura 5.2.5: Gráficas de resultados viga simplemente apoyada

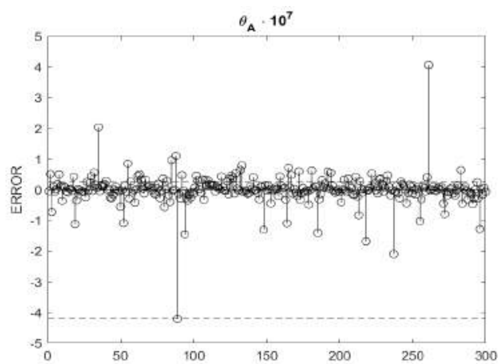


(a) Error (R_{Ay})

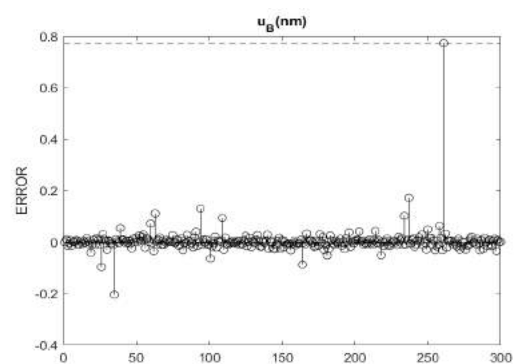


(b) Error (R_{By})

Figura 5.2.6: Gráficas de resultados viga simplemente apoyada

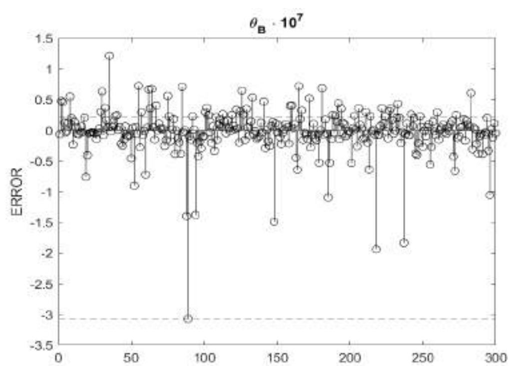


(a) Error (Ángulo en A)



(b) Error (Desplazamiento horizontal en B)

Figura 5.2.7: Gráficas de resultados viga simplemente apoyada



(a) Error (Ángulo en B)

Figura 5.2.8: Gráficas de resultados viga simplemente apoyada

	Error absoluto (Test)		
Salidas	Máximo	Mínimo	Media
$R_{Ax}(kN)$	0.0209	-0.0000032791	0.0022
$R_{Ay}(kN)$	0.3944	0.0000183	0.028
$R_{By}(kN)$	-0.3792	0.0000706	0.0283
$\theta_A \cdot 10^7$	-4.2095	0.0034	0.2605
$u_B(nm)$	0.7746	-0.0001	0.0181
$\theta_B \cdot 10^7$	-3.0701	0.0011	0.2288

Tabla 5.4: Errores absolutos en las predicciones de la red para el problema de la viga biapoyada usando el algoritmo de Levenberg-Marquardt

5.3. Resolución de ejemplos

Se va a definir un ejemplo de problema de cada tipo de los anteriores, para comparar con ejemplos reales el rendimiento que se obtiene con las redes neuronales, y la diferencia de usar un algoritmo de aproximación u otro. En la tabla 5.5 se presentan las entradas de ambos problemas. En las tablas 5.6 y 5.7 se ponen las soluciones para tres casos, solución con smmatlab por la teoría de Euler–Bernoulli y las soluciones dadas por las redes neuronales entrenadas en el capitulo anterior. Por último, en la figura 5.3.1, se representan las deformadas del problema de esfuerzos constantes obtenido por los tres métodos. La red neuronal entrenada por Levenberg-Marquardt y el smmatlab dan los mismos resultados y las deformadas se solapan.

Viga con esfuerzos constantes					
$L(m)$	$P(kN)$	$M_x(kNm)$	$M_z(kNm)$	$E(Gpa)$	$G(Gpa)$
10	4	3	3	210	78
Viga simplemente apoyada					
$L(m)$	$x_L(m)$	$P_x(kN)$	$P_y(kN)$	$E(Gpa)$	$G(Gpa)$
10	7	1.5	2	210	78

Tabla 5.5: Definición de los problemas de ejemplo

	$N(kN)$	$M_t(kNm)$	$M_f(kNm)$	$v(L)(nm)$	$\theta \cdot 10^9(L)$	$u(L)(nm)$	$\theta_x \cdot 10^7(L)$
Euler–Bernoulli	4.00	3.00	3.00	5.68	11.37	1.52	1.53
Descenso del gradiente	3.01	9.19	0.06	-7.70	5.43	3.19	7.14
Levenberg-Marquardt	4.00	3.00	3.00	5.68	11.38	1.52	1.52

Tabla 5.6: Soluciones ejemplo problema de viga con esfuerzos constantes

	$R_{Ax}(KN)$	$R_{Ay}(kN)$	$R_{By}(kN)$	$\theta_A(kN) \cdot 10^7$	$U_{Bx}(nm)$	$\theta_B \cdot 10^7$
Euler–Bernoulli	-1500.00	600.00	1400.00	-344.84	397.89	450.94
Descenso del gradiente	-1294.60	494.40	1456.40	-201.20	464.30	298.80
Levenberg-Marquardt	-1500.00	600.00	1400.00	-345.10	397.80	450.80

Tabla 5.7: Soluciones ejemplo problema de viga simplemente apoyada

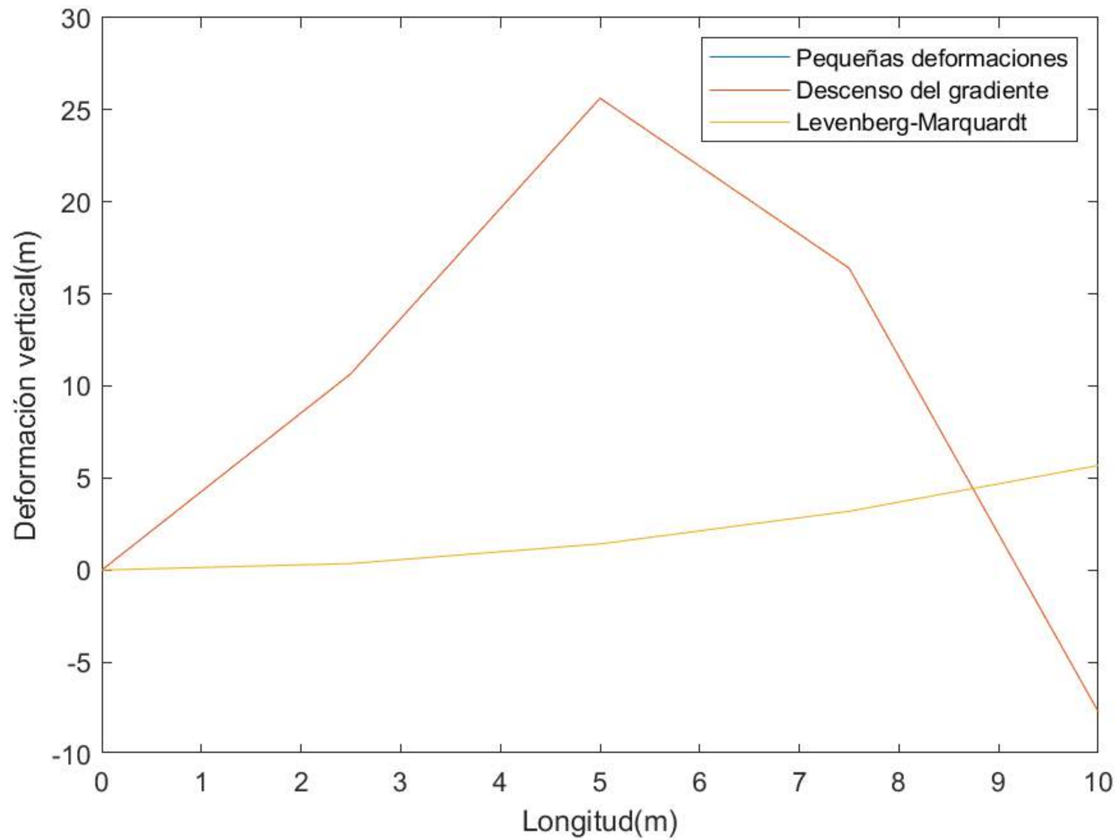


Figura 5.3.1: Deformadas obtenidas por los tres métodos para la viga empotrada sometida a esfuerzos constantes, las deformadas obtenidas con la red por Levenberg-Marquardt y por smmatlab se solapan

Capítulo 6

Resolución de un problema de varias cargas con una red neuronal

En capítulos previos se ha visto como las redes neuronales son capaces de aproximar problemas de vigas, en dichos problemas, se podía modificar el tamaño y material de la viga a placer, y la red neuronal captaba los matices de dicho cambio. En cuanto a las cargas, en el caso de la viga empotrada, se tenían 3 cargas, un momento torsor, un flector y una fuerza axial, siempre aplicadas en el mismo lugar. En la viga biapoyada, tan solo se contaba con una carga puntual, que esta sí que podía cambiar de punto de aplicación.

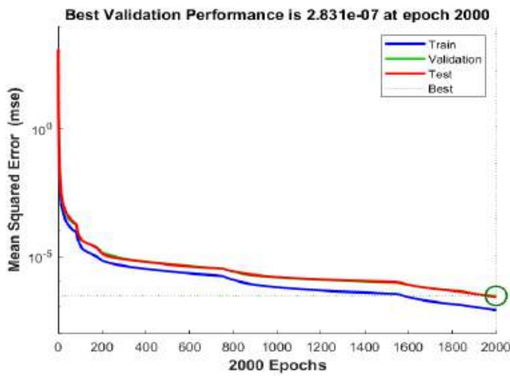
En esta última red neuronal, se cambia el enfoque, se establecerá una estructura concreta, en este caso una viga empotrada por su sencillez (8 metros de longitud y de acero), y se le colocarán hasta cuatro cargas, dos fuerzas puntuales y dos momentos, que podrán variar su punto de aplicación en toda la viga. De esta manera, se podrán obtener las soluciones de una estructura en miles de casos de carga distintos, de una manera rápida y efectiva. Los vectores de entrada y salida de la red son los que siguen:

- Entrada: $[x_{L1}, P_{x1}, P_{y1}, x_{L2}, P_{x2}, P_{y2}, x_{L3}, M_{x1}, M_{z1}, x_{L4}, M_{x2}, M_{z2}]$
- Salida: $[R_x, R_y, M_x, M_z, u(L), v(L), \theta_x(L), \theta_z(L)]$

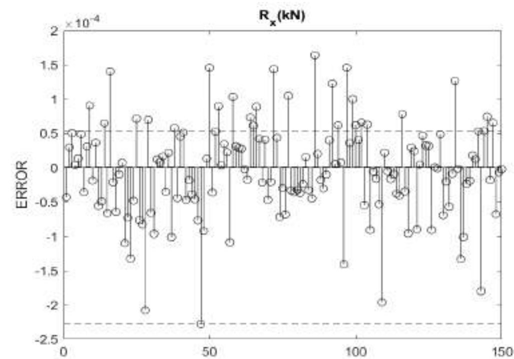
La arquitectura de la red, tamaño del dataset y parámetros de entrenamiento usados en el entrenamiento son los siguientes:

- Tamaño del dataset: 1000
- Neuronas de la capa oculta: 64
- Epoch máximas: 2000
- $\mu = 0.01$
- Decremento de μ: 0.1
- Incremento de μ: 10

Al igual que con los entrenamientos previos, se muestra la progresión del mse durante el entrenamiento, figura 6.0.1a, en este caso se observa un entrenamiento sostenido, con un mse que decrece mucho en la parte inicial y que luego desciende poco a poco alrededor del mínimo, además, las lineas de entrenamiento, validación y test se encuentran pegadas durante todas la epochs, lo que garantiza que no se esté produciendo un fenómeno de overfiting. A continuación, se muestras las gráfica del error para cada observación de salida. En las gráficas (figura 6.0.1b hasta 6.0.5a), se puede apreciar que los ordenes de magnitud oscilan entre 10^{-3} y 10^{-4}, siendo el error medio menor en los tres primeros outputs, aunque manteniéndose en valores más que aceptables en el resto.

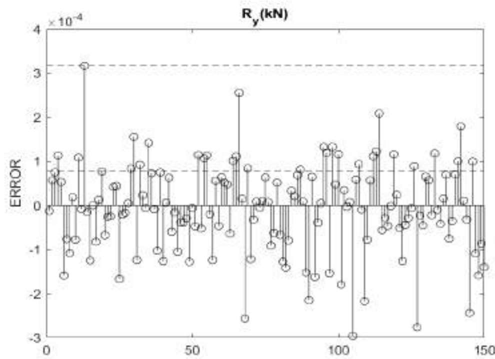


(a) Evolución del mse durante el entrenamiento

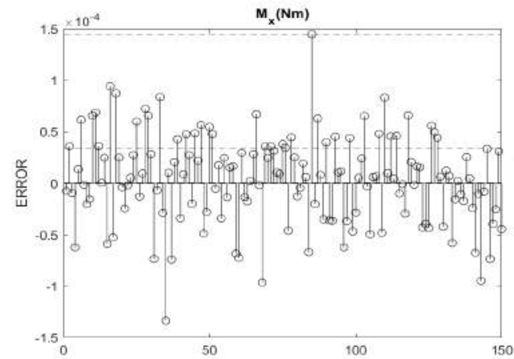


(b) Error (R_x)

Figura 6.0.1: Gráficas de resultados problema de varias cargas

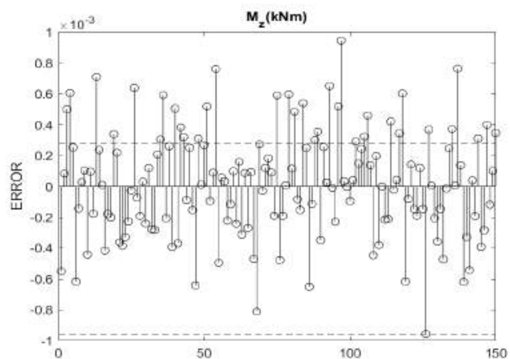


(a) Error (R_y)

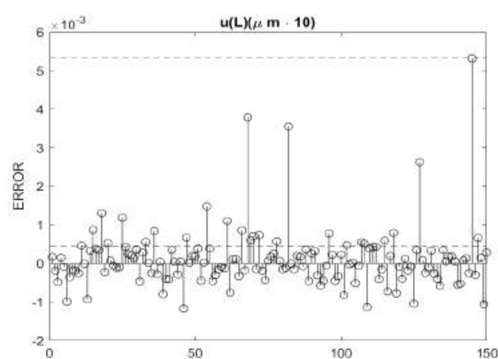


(b) Error (M_x)

Figura 6.0.2: Gráficas de resultados problema de varias cargas

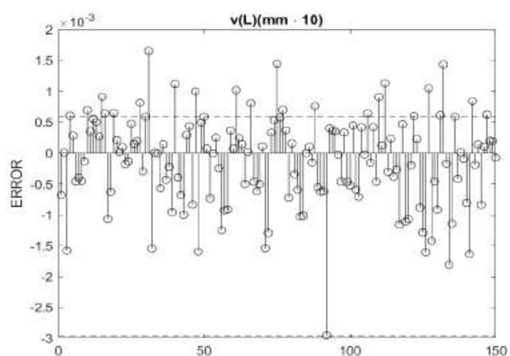


(a) Error (M_z)

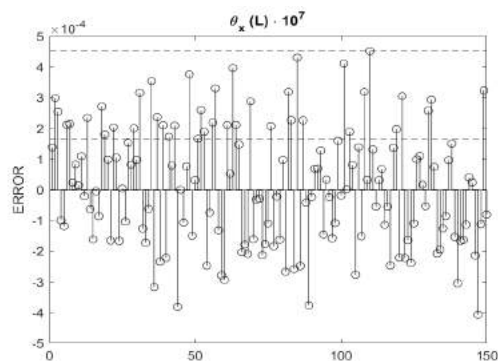


(b) Error ($u(L)$)

Figura 6.0.3: Gráficas de resultados problema de varias cargas

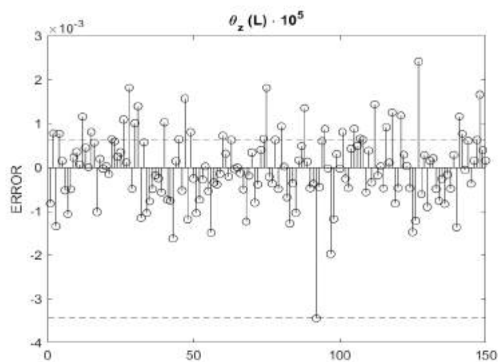


(a) Error ($v(L)$)



(b) Error ($\theta_x(L)$)

Figura 6.0.4: Gráficas de resultados problema de varias cargas



(a) Error ($\theta_z(L)$)

Figura 6.0.5: Gráficas de resultados problema de varias cargas

	Error absoluto (Test)		
Salidas	Máximo	Mínimo	Media
$R_x(kN)$	-0.0002	0.000000669	0.000054
$R_y(kN)$	0.0003	-0.000000196	0.0000783
$M_x(Nm)$	0.0001	0.000000547	0.0000344
$M_z(kNm)$	-0.001	-0.000000222	0.0002792
$u(L)(\mu m \cdot 10)$	0.0053	-0.000005045	0.0004591
$v(L)(mm \cdot 10)$	-0.003	-0.000001435	0.0005884
$\theta_x(L) \cdot 10^7$	0.0005	0.000000003	0.0001656
$\theta_z(L) \cdot 10^5$	-0.0034	-0.000000630	0.0006384

Tabla 6.1: Errores absolutos para el problema de varias cargas en el conjunto de test

A continuación, se mostrará una comparativa de como resuelve la red neuronal distintos casos de cargas no vistos nunca durante el entrenamiento, comparados con la resolución tradicional por la teoría de Euler-Bernoulli. Se introducirán 6 casos, mostrados en la tabla 6.2, que irán sumando cargas al problema, empezando desde el caso 1, donde sólo se aplica con una carga puntual, hasta el caso 6, de dos cargas puntuales y dos momentos. En la tabla 6.3, aparecen una seguida de otra las soluciones de los 6 casos por ambos métodos, y se comprueba la proximidad de los resultados obtenidos, que se extiende al segundo decimal.

Caso	x_{L1}	P_{x1}	P_{y1}	x_{L2}	P_{x2}	P_{y2}	x_{L3}	M_{x1}	M_{z1}	x_{L4}	M_{x2}	M_{z2}
1	4	1	1.5	0	0	0	0	0	0	0	0	0
2	4	1	1.5	6	0.5	1.75	0	0	0	0	0	0
3	0	0	0	0	0	0	2	1	1.5	0	0	0
4	0	0	0	0	0	0	2	1	1.5	8	0.5	1.75
5	4	1	1.5	0	0	0	8	1	1.5	0	0	0
6	4	1	1.5	6	0.5	1.75	8	1	1.5	2	0.5	1.75

Tabla 6.2: Datos de entrada de los 6 casos de carga

Caso	Método	$R_x(kN)$	$R_y(kN)$	$M_x(Nm)$	$M_z(kNm)$	$u(L)(\mu m \cdot 10)$	$v(L)(mm \cdot 10)$	$\theta_x(L) \cdot 10^7$	$\theta_z(L) \cdot 10^5$
1	EB	-1	-1.5	0	-6	1.5158	3.0315	0	4.5473
	RN	-1	-1.4998	-0.0002	-5.9963	1.507	3.0358	0.0008	4.5503
2	EB	-1.5	-3.25	0	-16.5	2.6526	10.1935	0	16.4839
	RN	-1.5	-3.2501	-0.0001	-16.4998	2.6524	10.1932	0.0008	16.4829
3	EB	0	0	-4	0	0	0	2.7587	0
	RN	-0.0003	0.0006	-4	0.0042	-0.0104	0.0049	2.7591	0.0056
4	EB	0	0	-4	0	0	0	2.7587	0
	RN	-0.0002	0.0006	-4	0.0048	-0.012	0.0065	2.7589	0.0058
5	EB	-0.5	-1.5	-8	-6	0.7579	3.0315	4.7292	4.5473
	RN	-0.5	-1.4999	-8	-5.9967	0.7503	3.0358	4.7289	4.5512
6	EB	-1.5	-3.25	-8	-16.5	2.6526	10.1935	4.7292	16.4839
	RN	-1.5001	-3.2501	-8	-16.4997	2.6526	10.1935	4.729	16.4841

Tabla 6.3: Comparación de las soluciones de los 6 casos dadas por la red neuronal (RN) y por la resolución convencional (EB)

Capítulo 7

Diseño de una red neuronal para una estructura genérica

7.1. Introducción

En todo desarrollo de *software* el desarrollo de estructura de datos es fundamental para la optimización tanto en tiempo como en tamaño. Además, resulta central en la legibilidad de código fuente. En el caso de redes neuronales es necesario definir bien el *input* ,pues es la definición del problema para la red y, por tanto, es central a la hora de que aprenda a resolver el problema adecuado.

7.2. Estructura fija

Hasta ahora se ha visto la definición de un problema concreto, una viga con ciertos apoyos en sus extremos, y con un número fijo de fuerzas aplicadas en ciertos puntos, que era el *input* de la red. Si el número de fuerzas (y su localización) fuera la curva continua que define el director de la viga, se obtenía la entrada genérica (infinita) para una estructura concreta (una viga con ciertos apoyos predeterminados). Como en el caso de métodos numéricos como diferencias finitas, volúmenes finitos o elementos finitos, no nos queda más remedio que discretizar la curva continua en un número finito de puntos en los que se puedan definir las fuerzas aplicadas. Es decir, dada $\mathcal{C} : \mathcal{I} \to \mathbb{R}^3$, se obtiene un conjunto de puntos $\mathcal{C}^h$ definidos como

$$\mathcal{C}^h := \left\{ \boldsymbol{x_i},\ \ \boldsymbol{x_i} \in \mathcal{C}\ \text{ y }\ |\boldsymbol{x}_{i+1} - \boldsymbol{x}_i| \leq h\ \ \forall i \in \mathcal{I}^h \right\} \tag{7.2.1}$$

donde $\mathcal{I}^h \subset \mathcal{I}$ tal que $\mathcal{C}(i) = \boldsymbol{x}_i\ \ \forall i \in \mathcal{I}^h$

La definición de (7.2.1) no es más que la definición de un mallado de $\mathcal{C}$ de tamaño h. Con este mallado discreto y finito queda definir el *input*, es decir, queda por definir en cada punto de ese mallado las condiciones de contorno, es decir, las fuerzas externas que actúan sobre él. Por lo que la estructura final del *input* debe ser una estructura que almacene:

- Los N puntos discretos en $\mathbb{R}^3$ del mallado $\mathcal{C}^h$. El número de variables necesarias para la definición son, por tanto, $3 \times N$ escalares.

- Para cada punto $\boldsymbol{x}_i \in \mathcal{C}^h$, las fuerzas externas que actúan sobre ese punto. Obsérvese que las fuerzas pueden ser fuerzas propiamente dichas o pares. El número de variables necesarias para la definición son, por tanto, $3 \times N + 3 \times N$ escalares.

Por lo que se tiene que el *input* de la red neuronal para una viga fija debe ser un estructura de datos de tamaño $3 \times N + (3 \times N + 3 \times N) = 9 \times N$. En el caso más general, estará formada por un *array* de tamaño $9 \times N$
Este razonamiento se puede generalizar para cualquier estructura concreta, obteniéndose así el *input* genérico (finito) para una estructura fija.

Conviene resaltar aquí que este procedimiento no permite introducir ciertas condiciones de contorno, como las debidas a los apoyos (restricciones en los desplazamientos) o a los enlaces (libertades en los desplazamientos entre dos secciones continuas). Esto se debe a que tal y como está definido el problema, estas condiciones forman parte de la estructura y se deben meter con la generalización del *input* para cualquier tipo de estructura.

7.3. Estructura genérica

Conviene hacer incapié aquí en el concepto de estructura a lo largo de este trabajo. Una estructura viene definida por un conjunto de vigas, un conjunto de conexiones entre vigas y un conjunto de apoyos. Todos estos conceptos se han definido previamente, viéndose los datos necesarios para su correcta definición computacional. Por lo que queda únicamente definir de forma ordenada todos ellos para generar una estructura genérica.
En primer lugar, observemos que las vigas definen dos nodos extremos y que los apoyos y enlaces, tal y como está definida la estructura únicamente pueden localizarse en alguno de estos nodos. Por tanto, al menos geométricamente, los nodos son los puntos más relevantes de la estructura. De hecho, dado un conjunto de nodos, $\mathcal{N}$, se pueden definir todas las posibles vigas conectadas entre ellos, así como sus apoyos y/o enlaces.
Por otro lado, el número de vigas, apoyos y enlaces y, por tanto, de nodos es indeterminado. Sin embargo, la red neuronal tiene un tamaño fijo de entrada, por lo que deberemos encarar el primer problema y la primera limitación. Resulta obvio que al estar fijo el número de entradas, deberemos limitarnos a estructuras con un número máximo de nodos. La limitación máxima de nodos es un problema normal y genérico en cálculo por ordenador. Pero queda pendiente ver cómo trabajar cuando se tienen menos nodos de los permitidos.
Una primera forma de resolver el problema con menos nodos del máximo es definir nodos ficticios sobre los reales, dando lugar a vigas y conexiones ficticias. Esto se puede hacer siempre definiendo vigas de longitud nula. Sin embargo, los recursos utilizados para resolver problemas pequeños serían desmedidos y no parece computacionalmente óptimo.
Otra forma es crear subredes de tamaño fijo con nodos máximos $\mathcal{N}_i^{max} = 2, 3, 4, ..., \mathcal{N}^{max}$ y recurrir a cada una de ellas dependiendo del problema a tratar. El inconveniente en este caso sería el coste de memoria, pues tendríamos que guardar $\mathcal{N}^{max} - 1$ redes neuronales de tamaño diferente en lugar de una sola. Para este tipo de problemas, la memoria no suele ser un problema en los ordenadores modernos, por lo que se primará el tiempo de cómputo y esta es la opción elegida.
Por lo tanto, una estructura de $\mathcal{N}_i^{max}$ nodos vendrá dada de la siguiente manera

- La posición en $\mathbb{R}^3$ de los nodos. El número de variables necesarias para su definición son

$$N_{tot}^{nodos} = 3 \times \mathcal{N}_i^{max} \tag{7.3.1}$$

- Todas las posibles vigas conectadas entre nodos. Cuando no existe viga entre dos nodos, se define una viga ficticia con longitud y rigideces nulas. El número de variables necesarias para su definición depende del modelo de viga que se quiera usar. Como ejemplo se puede usar una viga recta de material isótropo y sección circular constante, el numero de parámetros serían 3. Dos para describir el material, por ejemplo el modulo de Young y el coeficiente de Poisson, y uno para describir la sección, por ejemplo el radio.

$$N_{tot}^{vigas} = N_{var}^{vigas} \times ((\mathcal{N}_i^{max} - 1) + (\mathcal{N}_i^{max} - 2) + (\mathcal{N}_i^{max} - 3) + ... + 1) = N_{var}^{vigas} \cdot \frac{\mathcal{N}_i^{max} \cdot (\mathcal{N}_i^{max} - 1)}{2} \tag{7.3.2}$$

  donde N_{var}^{vigas} es el número de parámetros necesario para definir una viga. El vector de entrada además debería seguir un orden, para saber que parámetros corresponden con la viga que conecta que nodos. Por ejemplo se podría establecer que primero fuese el nodo 1 con el 2, luego el 1 con el 3, hasta el 1 con el i, seguido del 2 con el 3 y así hasta el último, que correspondería con la viga entre el nodo i-1 al i.

- Los apoyos en cada nodo. En caso de que no haya apoyo, se define un apoyo ficticio en el que se permiten todos los movimientos. El número mínimo de variables necesarias para su definición son 14. Las siete primeras variables corresponden a los bloqueos a movimientos de traslación y las 7 siguientes a las rotaciones. El número máximo de direcciones en las que se puede prohibir el movimiento son tres, y deben de ser ortogonales. Los primeros 6 parámetros sirven para indicar las coordenadas de los vectores en las direcciones que prohíben los dos primeros desplazamientos, que serán nulos en caso de estar libres, y la última variable tomará el valor de 0 si el movimiento en dirección ortogonal a los dos anteriores está permitido y 1 en caso de estar bloqueado. La misma dinámica es seguida para las rotaciones.

$$N_{tot}^{apoyos} = N_{var}^{apoyos} \times \mathcal{N}_i^{max} \tag{7.3.3}$$

- Las conexiones en los nodos. El número de variables mínimas necesarias para su definición son, al igual que con los apoyos, 14. Aunque en este caso, como lo normal es contar con conexiones rígidas, se cree que la estrategia más inteligente, en vez de reflejar las direcciones donde se bloquea el movimiento, usar las direcciones en las que se permite el movimiento relativo libre entre las vigas de la conexión.

$$N_{tot}^{conexiones} = N_{var}^{conexiones} \times n_{vigas} = N_{var}^{conexiones\ \mathcal{N}_i^{max}} \times \frac{\mathcal{N}_i^{max} \cdot (\mathcal{N}_i^{max} - 1)}{2} \tag{7.3.4}$$

  donde $N_{var}^{conexiones\ \mathcal{N}_i^{max}}$ es el máximo número de variables necesario para definir una conexión entre $\mathcal{N}_i^{max} - 1$ vigas, dado que este es el número máximo de vigas que se pueden conectar en un nodo en un problema de $\mathcal{N}_i^{max}$ nodos. Para definir una conexión entre n_{vigas} vigas es necesario definir los movimientos permitidos entre las secciones 2

a 2 para cada viga menos para una (para la última viga en el recuento los movimientos permitidos se pueden deducir del resto de emparejamientos). Por lo que el número de variables necesario es

$$N_{var}^{conexiones\ \mathcal{N}_i^{max}} = N_{var}^{conexiones\ 2} \cdot (n_{vigas} - 1) = N_{var}^{conexiones\ 2} \times (\mathcal{N}_i^{max} - 2) \quad (7.3.5)$$

donde $N_{var}^{conexiones\ 2}$ es el número de variables para definir una conexión entre dos vigas (`basicConnection` en la nomenclatura de smMatlab), que es constante.

Finalmente, sustituyendo (7.3.5) en (7.3.4)

$$N_{tot}^{conexiones} = \frac{N_{var}^{conexiones\ 2}}{2} \times \mathcal{N}_i^{max} \cdot (\mathcal{N}_i^{max} - 1) \cdot (\mathcal{N}_i^{max} - 2) \quad (7.3.6)$$

Finalmente, junto a toda esta definición de la estructura hay que definir, por supuesto, las condiciones de contorno dadas por las fuerzas externas y que vienen dadas por lo visto en el apartado anterior y dependen del número de nodos (vigas) máximo de la estructura y del número de puntos de fuerzas externas en las vigas, N. Suponiendo N constante para todas las vigas, el tamaño de las fuerzas externas es:

$$N_{tot}^{fe} = N_{var}^{fe} \times n_{vigas} = (9 \times N) \times \frac{\mathcal{N}_i^{max} \cdot (\mathcal{N}_i^{max} - 1)}{2} \quad (7.3.7)$$

El número de variables escalares necesarias para definir el *input* de una estructura genérica es simplemente la suma de (7.3.1), (7.3.2), (7.3.3), (7.3.6) y (7.3.7). Obsérvese que el tamaño del *input* depende linealmente de la discretización de las vigas, pero más importante todavía, depende cúbicamente del número de nodos. En los gráficos de la figura 7.3.1 se muestra el crecimiento del número de entradas según se aumentan los nodos de la estructura. En él se puede ver claramente como la contribución más importante es la debida a las conexiones, a causa del crecimiento cúbico ya comentado.

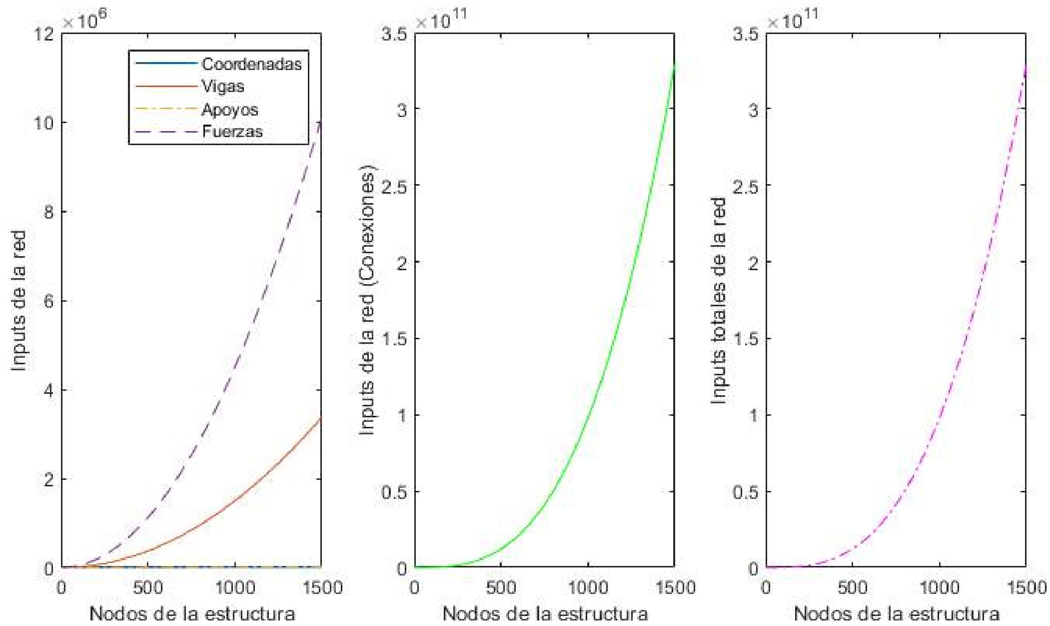


Figura 7.3.1: Evolución del número de entradas de la red necesario para describir la estructura genérica en función de los nodos de ésta. En el gráfico de la izquierda se representan cuatro rectas de evolución, que representan el número de inputs debidos a las coordenadas de los nodos, la definición de las vigas, apoyos y fuerzas. En la gráfica central solo se representan aquellos inputs aportados por la definición de las conexiones. En la último gráfica se tiene la representación de la suma de todos los inputs anteriores.

7.3.1. Salida de datos

Añadir que una red neuronal no solo necesita una estructura de datos para definir las entradas, sino también las salidas o las soluciones a los problemas. En este caso son más sencillas de describir, puesto que lo más lógico es usar 12 salidas por cada nodo y viga que conecte dicho nodo, tal y como se calcula en la ecuación 7.3.8. Las 6 primeras almacenando las fuerzas internas y momentos en la sección y las otras 6 para los desplazamientos y giros en las tres direcciones del espacio.

$$N_{tot}^{sal} = 12 \times \mathcal{N}_i^{max} \cdot (\mathcal{N}_i^{max} - 1) \tag{7.3.8}$$

Capítulo 8

Conclusiones y lineas futuras

8.1. Conclusiones

La selección de la arquitectura e hiperparámetros de una red neuronal no es un tema trivial, los miles de artículos escritos sobre el tema y las centenares de técnicas y modificaciones de las técnicas para conseguir que las redes neuronales aprendan es un claro síntoma de ello. De hecho, actualmente es común el uso de redes pre-entrenadas, a las que simplemente se adaptan para los datos deseados, ya que facilita el trabajo y evita muchos rompecabezas. En este caso, no se hizo uso de ninguna de estas redes, definiendo todos los detalles de las redes y los entrenamientos desde el inicio. A pesar de que se usaron problemas y modelos sencillos, esta decisión marcó el devenir del proyecto, causando un gran estancamiento. En un principio se utilizó el método de gradiente para resolver el problema de la red neuronal, de hecho, en primera instancia, fue el solver Adam el que se escogió para entrenar a los modelos, el problema estuvo en que no había forma de que las redes aprendiesen, ni aumentando el tamaño del dataset, el número de neuronas, modificando la función de activación, alterando la tasa de aprendizaje, añadiendo regularización, era imposible obtener un modelo válido para la tarea, y que sus predicciones fuesen algo más precisas que un generador de números aleatorios. No fue hasta el hallazgo del artículo de Kaustav Sarkar que se decidió probar con el algoritmo ya mencionado de Levenberg-Marquardt, momento en el que el entrenamiento se dejó de estancar en un mínimo local y empezó a encontrar el mínimo global deseado. En general, en el campo de las redes neuronales, no existe ningún método que garantice que la red va a aprender y la mayoría de aprendizaje se basa en la experiencia. A la conclusión que se quiere llegar con esto no es más que el Machine Learning es un campo yermo, lo que lo hace una de las disciplinas más emocionantes del futuro, y a esto se le suman los sorprendentes resultados que se están obteniendo con modelos complejos de deep learning como GPT-3 o DALL-E (https://openai.com/research/).

Otra de las conclusiones importantes a las que se ha querido llegar en este trabajo es la diferenciación entre la minimización de problemas convexos y no convexos. La aproximación polinómica es un problema convexo, por lo que cualquier mínimo local es global. Los problemas convexos han sido y son extensamente estudiados, por la característica mencionada de los mínimos locales, lo que aseguran la optimabilidad y hay una extensa teoría matemática construida. Aunque no se puede garantizar encontrar la solución del problema, como ocurrió

en un primer momento con la red neuronal, existen una gran cantidad de algoritmos con una gran base matemática para su resolución. En cuanto a las redes neuronales la cosa no es igual, en principio no se sabe si el problema es convexo, pero lo más probables es que no lo sea, y es por eso por lo que no conocemos si quiera donde vamos con el aprendizaje. Esta diferenciación supone un punto muy negativo de las redes neuronales, pero claro, su potencial, debido a su probado éxito en variados problemas, supone un punto positivo tan grande que no va a permitir su desaparición en detrimento de los problemas del primer tipo.

Por último, realizar un comentario sobre la estructura genérica, lo óptimo hubiese sido incluir los resultados de una red entrenada que usase dicha estructura y que pusiese en práctica lo propuesto, sin embargo, un simple estructura de 6 nodos requiere de una red constituida por 20951 entradas, sumado a la lentitud del lenguaje de programación usado, Matlab, supone que su elaboración se escape al alcance de este trabajo y requiera el uso de otros lenguajes más especializados como C++. Un detalle importante de resaltar, es la expresión 7.3.6, en las que se observa el crecimiento cúbico del número de entradas de la red con el número de nudos, y lo que implica es que los vectores de entrada aumentarán rápidamente conformen aumente el número de nudos. Dicha expresión corresponde con el aporte de entradas referidos a las conexiones, lo que significa que en caso de querer resolver estructuras donde todas las conexiones sean la misma, o solo haya unas pocas opciones, como pórticos y celosías, se podría decidir realizar redes más compactas.

8.2. Lineas futuras

A continuación se realizara una pequeña lista con las posibles lineas futuras de investigación que podrían complementar y continuar este trabajo.

- Generar datasets y entrenar redes más grandes, hasta miles o millones de nudos, usando la estructura genérica planteada y lenguajes de programación no interpretados como C++.
- Realizar modelos con vigas no rectas, secciones variables y materiales no isótropos.
- Usar datos de entrenamiento con ruido, captados de estructuras reales con sensorización, entrenar con un parámetro de regularización elevado para que sea capaz de ignorar el ruido de las observaciones.

Apéndice A

Código fuente

```
%% CODIGO 1 %%

%cte_combined_stress: genera las matrices X e Y de inputs y outputs
%de una red neuronal con problemas de vigas empotradas.
%Estructura de los datos:
%X(i, :) = [L, P, Mx, Mz, E, G]
%Y(i, :) = [N, Mt, Mfz, v(0.25L), theta(0.25L), v(0.5*L), theta(0*5L),
%v(0.75L), theta(0.75L), v(L), theta(L), u(L), thetax(L)]
%Autor/es: Victor Pastor

%numero de problemas a generar
N = 2000;

%Inicializar X e Y de entrenamiento
X = zeros(N, 6);
Xmod = zeros(N, 6); %[L, P, Mx, Mz, 1/E, 1/G]
Y = zeros(N, 13);

for i=1:N
    L = rand*16 + 4; %Longitud de la viga entre 4 y 20 metros
    E = rand*150 + 60; %Modulo de Young en Gpa, entre aluminio (60) y
                       % acero (210)
    nu = rand*0.1 + 0.25; %nu: 0.25 - 0.35
    G = E/(2*(1 + nu));
    bm = csBeam([[0.0 0.0 0.0]' [L 0.0 0.0]'],...
               {'Mat', [8e3, E*1e9, nu]}, {@cirSec, {20e-2}});
    %viga de sección cte. R = 20cm
    P = rand*8000 + 2000; %Entre 2000 y 10000N
    bF=[L;0.0;0.0;P;0.0;0.0];
    Mx = rand*8000 + 2000; %Entre 2000 y 10000Nm
    Mz = rand*8000 + 2000; %Entre 2000 y 10000Nm
    bM=[L;0.0;0.0;Mx;0.0;Mz];
    bempty = [0.0;0.0;0.0;0.0;0.0;0.0];
    X(i, :) = [L, P/1000, Mx/1000, Mz/1000, E, G];
    Xmod(i, :) = [L, P/1000, Mx/1000, Mz/1000, 1/E, 1/G];
    s1 =  fixed([0.0;0.0;0.0]); %Empotramiento
    s=structure({bm},{},{s1});
    sm=smProblem(s,bF,bempty,bM);
```

```
    sm.solve
    sm.structure.beams{1,1}.setDeformationLaw()
    [U, R] = sm.structure.beams{1,1}.getDisplacements(0.25*L);
    %U dezplazamientos, R giros a 0,25
    vL1 = U(2);
    thetaL1 = R(3);
    [U, R] = sm.structure.beams{1,1}.getDisplacements(0.5*L);
    %U dezplazamientos, R giros en el medio
    vL2 = U(2);
    thetaL2 = R(3);
    [U, R] = sm.structure.beams{1,1}.getDisplacements(0.75*L);
    %U dezplazamientos, R giros a 0,75
    vL3 = U(2);
    thetaL3 = R(3);
    [U, R] = sm.structure.beams{1,1}.getDisplacements(L);
    %U dezplazamientos, R giros en el extremo
    uL = U(1);
    vL4 = U(2);
    thetaL4 = R(3);
    thetaxL = R(1);
    v_modL1 = vL1.*1e4;
    v_modL2 = vL2.*1e4;
    v_modL3 = vL3.*1e4;
    v_modL4 = vL4.*1e4;
    u_modL = uL.*1e6;
    theta_modL1 = thetaL1.*1e5;
    theta_modL2 = thetaL2.*1e5;
    theta_modL3 = thetaL3.*1e5;
    theta_modL4 = thetaL4.*1e5;
    theta_modxL = thetaxL.*1e4;
    Y(i, :) = [sm.structure.beams{1, 1}.intF(1)/1000, ...
               sm.structure.beams{1, 1}.intM(1)/1000, ...
               sm.structure.beams{1, 1}.intM(3)/1000, ...
               v_modL1, theta_modL1, v_modL2, ...
               theta_modL2, v_modL3, theta_modL3, v_modL4, ...
               theta_modL4, u_modL, theta_modxL];
end
```

```
%% CODIGO 2 %%

%example_generator_ss: genera las matrices X e Y de inputs y outputs
%de una red neuronal con problemas de vigas simnplemente apoyadas.
%Estructura de los datos:
%X(i, :) = [L, Xl, Px, Py, E, G]
%Y(i, :) = [RAx, RAy, RBy, thetaA, UBx, thetaB]
%Autor/es: Victor Pastor


%numero de problemas a generar
n = 2000;

%Inputs y outputs de la red neuronal: 4 y 6
%Inicializar X e Y
X = zeros(n, 6);
```

```
Y = zeros(n, 6);

for i = 1:n
    %longitud aleatoria entre 4 y 20 metros
    L = rand*16 + 4;
    E = rand*150 + 60; %Modulo de Young en Gpa, entre aluminio (60)
    % y acero (210)
    nu = rand*0.1 + 0.25; %nu: 0.25 - 0.35
    G = E/(2*(1 + nu));
    bm = csBeam([[0.0 0.0 0.0]' [L 0.0 0.0]'], ...
                {'Mat', [8e3, E*1e9, nu]}, {@cirSec, {20e-2}});
    %viga de sección cte. R = 20cm

    %Definición de una fuerza con componentes X e Y en un punto arbitrario

    Xl = rand*L;
    Px = rand*2000;
    Py = -rand*2000; %Entre 0:2 kN
    bF=[Xl;0.0;0.0;Px;Py;0.0];
    X(i, :) = [L, Xl, Px/1000, abs(Py/1000), E, G];

    s1 =  pinnedRx([0.0;0.0;0.0]);
    s2 = roller1([L;0.0;0.0], [1.0 0.0 0.0]');
    s=structure({bm},{},{s1,s2});
    sm=smProblem(s,bF);
    sm.solve
    Y(i, :) = [sm.ueF(1:2), sm.ueF(5), ...
               sm.structure.beams{1}.rVec(3, 1), ...
               sm.structure.beams{1}.uVec(1, 2), ...
               sm.structure.beams{1}.rVec(3, 2)];
    Y(i, 4) = Y(i, 4).*1e7; %angulo x10^7
    Y(i, 5) = Y(i, 5).*1e9; %desp en nm
    Y(i, 6) = Y(i, 6).*1e7; %angulo x10^7
end
```

```
%% CODIGO 3 %%

%ml_polynomial: Función de entrenamiento con el método de aproximación
%polinómica
%Autor/es: David Portillo y Víctor Pastor

function [YsolH,Phi,PhiH,Xalpha,objFunalpha,Yapprox, Yapprox_test] = ...
         ml_polynomial(X,Y, X_test, grado)

gr = grado;
ndata = size(X,1); %numero de ejemplos de entrenamiento

nx = size(X,2); %variables de entrada
ny = size(Y,2); %variables de salida

[Phi, DPhi, PhiH, DPhiH] = generatePolyBasis(nx,gr);

nphi = length(Phi); %numero de funciones de base
```

```
alpha = sym('alpha',[1,nphi*ny]);
%numero de parametros alpha que multiplican a las funciones de base

objFunction = 0.0;

vPhiH = zeros(1,nphi);

for i=1:ndata

    auxdata = {};
    for kn = 1:nx
        auxdata = [auxdata,X(i,kn)];
    end
    mydata = struct('x',auxdata);

    for k=1:nphi
        vPhiH(k) = double(PhiH{k}(mydata.x));
    end
    for j = 1:ny

        k0 = (j-1)*nphi;
        approxYj = 0.0;
        for k=1:nphi
            approxYj = approxYj + alpha(k0+k)*vPhiH(k);
        end

        diff = approxYj - Y(i,j);

        objFunction = objFunction + diff*diff;
    end

end

objFunctionH = matlabFunction(objFunction,'Vars',{alpha});

alpha0 = zeros(1,nphi*ny);
options = optimoptions('fmincon','MaxFunctionEvaluations',10000, ...
                       'MaxIterations', 1000, 'StepTolerance', 1e-14);
[Xalpha,objFunalpha] = fmincon(objFunctionH,alpha0, ...
                               [],[],[],[],[],[],[],options);

x = sym('x',[1,nx]);
for j=1:ny
    Ysol(j) = simplify(dot(Xalpha((j-1)*nphi+1:j*nphi),Phi));
    YsolH{j} = matlabFunction(Ysol(j),'Vars',x);
end


for i=1:ndata

    auxdata = {};
    for kn = 1:nx
        auxdata = [auxdata,X(i,kn)];
    end
```

```
    mydata = struct('x',auxdata);

    for j=1:ny
        Yapprox(i,j) = YsolH{j}(mydata.x);
    end
end

ndata_test = size(X_test,1);

for i=1:ndata_test

    auxdata = {};
    for kn = 1:nx
        auxdata = [auxdata,X_test(i,kn)];
    end
    mydata = struct('x',auxdata);

    for j=1:ny
        Yapprox_test(i,j) = YsolH{j}(mydata.x);
    end
end

for j=1:ny
    figure(j);
    hold on;
    plot(1:ndata,Yapprox(:,j));
    plot(1:ndata,Y(:,j));
    legend('approx','data');
    filename = "training_" + string(j) + ".png";
    saveas(gcf, filename);
end

end
```

```
%% CODIGO 4 %%

%train_routine_pol: Script de llamada a la función de entrenamiento
%polinómica, con selección de parametros y datasets
%Autor/es: Víctor Pastor

mod = false;
%Carga de los datasets de entrenamiento y test
obj = matfile('simply_supported\datasets\ss_def.mat');
if mod
    X = obj.Xmod;
else
    X = obj.X;
end
Y = obj.Y;
ntr = 10;
%%%%%%%%%%%%%%%%
% Combinados:
% Combinados (mod): 210
% SS:
```

```
%%%%%%%%%%%%%%%%
N = round(ntr/0.85); %Observaciones tr + test
X = X(1:N, :);
Y = Y(1:N, :);
ind = randperm(N);
indTr = ind(1:round(0.85*N));
indT = ind(round(0.85*N)+1:N);
X_tr = X(indTr, :);
Y_tr = Y(indTr, :);
X_test = X(indT, :);
Y_test = Y(indT, :);

grado = 5;
%%%%%%%%%%%%%%%%
% Combinados:
% Combinados (mod): 4
% SS:
%%%%%%%%%%%%%%%%
[YsolH,Phi,PhiH,Xalpha,objFunalpha,Yapprox,Yapprox_test] = ...
                        ml_polynomial(X_tr,Y_tr,X_test, grado);
perform = mse(Y_tr, Yapprox) %#ok<NOPTS>

error = (Y_test - Yapprox_test)';
mean_error = sum(abs(error), 2)/(0.15*N);
max_min = zeros(size(Y,2), 2);
for i=1:size(Y,2)
    max_min(i ,1) = sign(error(i, abs(error(i, :)) == ...
                    max(abs(error(i, :))))) * max(abs(error(i, :)));
    max_min(i ,2) = sign(error(i, abs(error(i, :)) == ...
                    min(abs(error(i, :))))) * min(abs(error(i, :)));
end
```

```
%% CODIGO 5 %%

%network_train_gdm: Función de entrenamiento con el método del
%descenso del gradiente con momento
%Autor/es: Víctor Pastor

function [Yapprox, tr, net] = network_train_gdm(X,Y, hu, ...
                                                alpha, lamb, ...
                                                val_pat, m)


%Definicion de la red
net = feedforwardnet(hu, 'traingdm');

%Parametros de entrenamiento
net.trainParam.max_fail = val_pat;
net.trainParam.lr = alpha;
net.performParam.regularization = lamb;
net.trainParam.mc = m;
net.trainParam.epochs = 2000;

net.performParam.normalization = 'none'; % 'standard'
```

```
[net, tr] = train(net,X,Y); %Entrenamiento


plotperform(tr)

Yapprox = net(X);
end
```

```
%% CODIGO 6 %%

%train_routine_gdm: Script de llamada a la función de entrenamiento
%con descenso del gradiente, con selección de parametros y datasets
%Autor/es: Víctor Pastor

%Carga de los datasets de entrenamiento, validacion y test
obj = matfile('simply_supported\datasets\ss_def.mat');
X = obj.X;
Y = obj.Y;
N = 100; %Observaciones tr + val + test
X = X(1:N, :)';
Y = Y(1:N, :)';

setdemorandstream(5)

hidden_units = 16; %Neuronas de la capa oculta
val_pat = 1000;
alpha = 0.0000001; %Tasa de aprendizaje
lambda = 0.1;  %Regularizacion
m = 0.6; %Momento


[Yapprox, tr, net] = network_train_gdm(X,Y, hidden_units, alpha, ...
                                       lambda, val_pat, m);
perf = perform(net,Yapprox,Y) %#ok<NOPTS>

Y_test = Y(:, tr.testInd);
Yapprox_test = Yapprox(:, tr.testInd);

error = Y_test - Yapprox_test;
mean_error = sum(abs(error), 2)/(0.15*N);
max_min = zeros(size(Y,1), 2);
for i=1:size(Y,1)
    max_min(i ,1) = sign(error(i, abs(error(i, :)) == ...
                    max(abs(error(i, :))))) * max(abs(error(i, :)));
    max_min(i ,2) = sign(error(i, abs(error(i, :)) == ...
                    min(abs(error(i, :))))) * min(abs(error(i, :)));
end
```

```
%% CODIGO 7 %%

%network_train_lm: Función de entrenamiento con el método de
%Levenberg - Marquardt
%Autor/es: Víctor Pastor

```

```
function [Yapprox, net, tr] = network_train_lm(X,Y, hu, mu, val_pat, ...
                                                  mu_decrease, mu_increase)

%Definicion de la red
net = feedforwardnet(hu, 'trainlm');
%net.layers{1}.transferFcn = 'poslin';

net.trainParam.max_fail = val_pat;
net.trainParam.mu = mu;
net.trainParam.mu_dec = mu_decrease;
net.trainParam.mu_inc = mu_increase;
net.trainParam.epochs = 3000;

[net, tr] = train(net,X,Y); %Entrenamiento


plotperform(tr)

Yapprox = net(X);
end
```

```
%% CODIGO 8 %%

%train_routine_gdm: Script de llamada a la función de entrenamiento
%con Levenberg - Marquardt, con selección de parametros y datasets
%Autor/es: Víctor Pastor

%Carga de los datasets de entrenamiento, validacion y test
obj = matfile('simply_supported\datasets\ss_def.mat');
X = obj.X;
Y = obj.Y;
N = 500; %Observaciones tr + val + test
X = X(1:N, :)';
Y = Y(1:N, :)';

%setdemorandstream(3)

hidden_units = 32; %Neuronas de la capa oculta
mu = 0.1; %LM parametro de aprendizaje
validation_patience = 1000;
mu_decrease = 0.1; %Decremento de mu
mu_increase = 10;  %Incremento de mu

[Yapprox, net, tr] = network_train_lm(X,Y, hidden_units, mu, ...
                                      validation_patience, ...
                                      mu_decrease, mu_increase);
perf = perform(net,Yapprox,Y) %#ok<NOPTS>

Y_test = Y(:, tr.testInd);
Yapprox_test = Yapprox(:, tr.testInd);

error = Y_test - Yapprox_test;
mean_error = sum(abs(error), 2)/(0.15*N);
max_min = zeros(size(Y,1), 2);
```

```
for i=1:size(Y,1)
    max_min(i ,1) = sign(error(i, abs(error(i, :)) == ...
                    max(abs(error(i, :))))) * max(abs(error(i, :)));
    max_min(i ,2) = sign(error(i, abs(error(i, :)) == ...
                    min(abs(error(i, :))))) * min(abs(error(i, :)));
end
```

Apéndice B

Planificación y presupuesto

En el siguiente anexo se incluirán las distintas tareas de planificación que se han llevado a cabo durante el proyecto. Para simplificar el proceso, se divide su realización en 25 subtareas más sencillas, a las que se le aporta una duración temporal aproximada, asignando una carga diaria de 4 horas. En el diagrama de Gantt de la siguiente página se ven reflejadas dichas tareas, así como su relación temporal y causal. Además, las 25 subtareas se agrupan en seis tareas más generales: Gestión del proyecto, bibliografía, documentación, desarrollo de modelos, análisis de resultados y defensa del TFG.

Seguido del diagrama de Gantt, se incluyen dos gráficos generados por Microsoft Proyect que resumen los gastos del proyecto en función de los recursos usados, en este caso, humanos y tecnológicos, y en función de las 8 tareas generales en las que se engloba el proyecto.

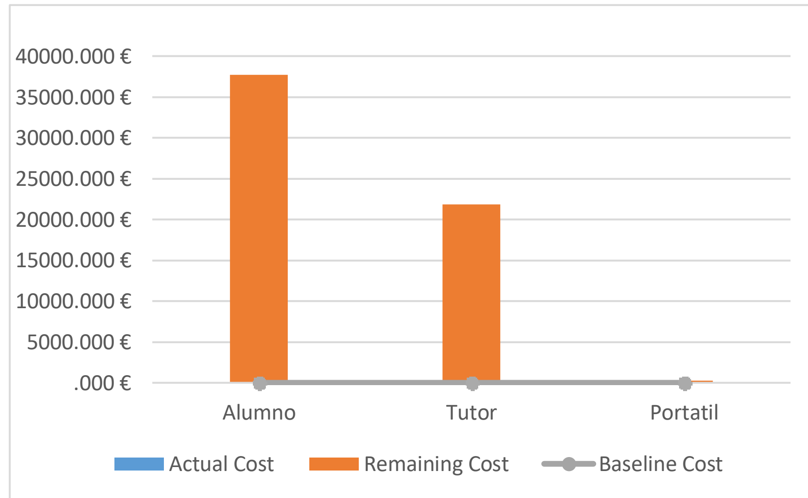
Por último, se incluye un presupuesto detallado del proyecto, en el que se incluyen las horas de trabajo destinadas a cada tarea, seguida de los recursos usados, y con el precio por recurso ligado a dichas horas de trabajo. El coste horario de un alumno de grado se fija en 40 euros la hora, el del tutor en 70 euros. Además se va a usar un portátil, que se amortiza el 26 % anual, lo que desemboca en un coste horario de 0,30€/h.

ID	Task Mode	Task Name	Duration	Start	Finish	Predecessors
1		**Gestión del proyecto**	**106 days**	**Mon 07/09/20**	**Mon 01/02/21**	
2		Discusión de los pobibles temas a tratar	1 day	Mon 07/09/20	Mon 07/09/20	
3		Investigación y selección del tema	15 days	Tue 08/09/20	Mon 28/09/20	2
4		Definición de objetivos	6 days	Tue 29/09/20	Wed 09/12/20	3
5		Planificación	5 days	Thu 10/12/20	Mon 01/02/21	4
6		**Bibliografía**	**141 days**	**Thu 01/10/20**	**Thu 15/04/21**	
7		Lectura "Machine Learning: A probabilistic persepective, Kevin P. Murphy"	25 days	Thu 01/10/20	Wed 04/11/20	3
8		Busqueda y selección de bibliografía	7 days	Thu 05/11/20	Fri 13/11/20	7
9		Lectura bibliografía	12 days	Mon 16/11/20	Tue 01/12/20	8
10		Lectura documentación, DL Toolbox Matlab	8 days	Tue 19/01/21	Thu 28/01/21	18;13
11		Ampliación de bibliografía y hallazgo del algoritmo de Levenberg-Marquardt	25 days	Fri 12/03/21	Thu 15/04/21	24
12		**Documentación**	**161 days**	**Wed 02/12/20**	**Wed 14/07/21**	
13		Redacción parte teórica de la memoria	34 days	Wed 02/12/20	Mon 18/01/21	9
14		Inclusión de modelos y resultados en la memoria	20 days	Thu 06/05/21	Wed 02/06/21	25
15		Revisión de la memoria	15 days	Thu 03/06/21	Wed 23/06/21	14
16		Redacción resumen e introducción	15 days	Thu 24/06/21	Wed 14/07/21	15
17		**Desarrollo de modelos**	**102 days**	**Fri 11/12/20**	**Mon 03/05/21**	
18		Desarrollo primeros modelos con la librería PMTK	10 days	Fri 11/12/20	Thu 24/12/20	7
19		Definición de los problemas y entradas y salidas de los	10 days	Fri 25/12/20	Fri 15/01/21	18
20		Programación scripts para la generación de datasets	16 days	Fri 29/01/21	Fri 19/02/21	10
21		Entrenamiento de modelos y generación de resultados I	12 days	Mon 22/02/21	Tue 09/03/21	20
22		Entrenamiento de modelos y generación de resultados II	12 days	Fri 16/04/21	Mon 03/05/21	11
23		**Análisis de resultados**	**41 days**	**Wed 10/03/21**	**Wed 05/05/21**	
24		Análisis de resultados I	2 days	Wed 10/03/21	Thu 11/03/21	21
25		Análisis de resultados II	2 days	Tue 04/05/21	Wed 05/05/21	22
26		Entrega TFG	0 days	Mon 06/09/21	Mon 06/09/21	16
27		**Defensa TFG**	**20 days**	**Mon 06/09/21**	**Fri 01/10/21**	
28		Preparación de la presentación	20 days	Mon 06/09/21	Fri 01/10/21	26
29		Defensa TFG	0 days	Fri 01/10/21	Fri 01/10/21	28

RESOURCE COST OVERVIEW

COST STATUS

Cost status for work resources.



COST DETAILS

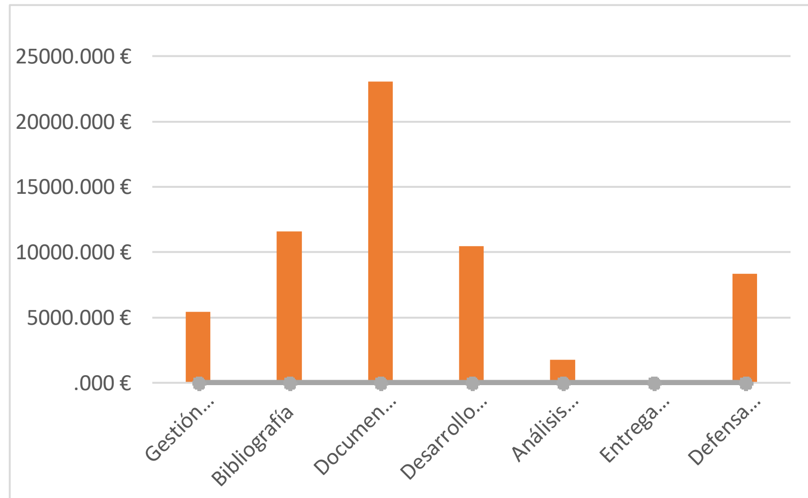
Cost details for all work resources.

Name	Standard Rate	Regular Work	Remaining Cost
Alumno	40,00 €/hr	944 hrs	37.760,00 €
Tutor	70,00 €/hr	312 hrs	21.840,00 €
Portatil	0,30 €/hr	964 hrs	289,20 €

TASK COST OVERVIEW

COST STATUS

Cost status for top-level tasks.



COST DETAILS

Cost details for all top-level tasks.

Name	Fixed Cost	Actual Cost	Remaining Cost	Cost	Baseline Cost	Cost Variance
Gestión del proyecto	0,00 €	0,00 €	5.424,00 €	5.424,00 €	0,00 €	5.424,00 €
Bibliografía	0,00 €	0,00 €	11.612,40 €	11.612,40 €	0,00 €	11.612,40 €
Documentación	0,00 €	0,00 €	23.080,40 €	23.080,40 €	0,00 €	23.080,40 €
Desarrollo de modelos	0,00 €	0,00 €	10.463,60 €	10.463,60 €	0,00 €	10.463,60 €
Análisis de resultados	0,00 €	0,00 €	1.764,80 €	1.764,80 €	0,00 €	1.764,80 €
Entrega TFG	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Defensa TFG	0,00 €	0,00 €	8.344,00 €	8.344,00 €	0,00 €	8.344,00 €

Gestión del proyecto	**200 hrs**	**106 days**	**Mon 07/09/20**	**Mon 01/02/21**	**5.424,00 €**
Discusión de los pobibles temas a tratar	12 hrs	1 day	Mon 07/09/20	Mon 07/09/20	441,20 €
Alumno	4 hrs		Mon 07/09/20	Mon 07/09/20	160,00 €
Tutor	4 hrs		Mon 07/09/20	Mon 07/09/20	280,00 €
Portatil	4 hrs		Mon 07/09/20	Mon 07/09/20	1,20 €
Investigación y selección del tema	120 hrs	15 days	Tue 08/09/20	Mon 28/09/20	2.418,00 €
Alumno	60 hrs		Tue 08/09/20	Mon 28/09/20	2.400,00 €
Portatil	60 hrs		Tue 08/09/20	Mon 28/09/20	18,00 €
Definición de objetivos	48 hrs	6 days	Tue 29/09/20	Wed 09/12/20	1.764,80 €
Alumno	16 hrs		Tue 29/09/20	Fri 02/10/20	640,00 €
Tutor	16 hrs		Tue 29/09/20	Fri 02/10/20	1.120,00 €
Portatil	16 hrs		Tue 29/09/20	Wed 09/12/20	4,80 €
Planificación	20 hrs	5 days	Thu 10/12/20	Mon 01/02/21	800,00 €
Alumno	20 hrs		Thu 10/12/20	Mon 01/02/21	800,00 €
Bibliografía	**596 hrs**	**141 days**	**Thu 01/10/20**	**Thu 15/04/21**	**11.612,40 €**
Lectura "Machine Learning: A probabilistic persepective, Kevin P. Murphy"	180 hrs	25 days	Thu 01/10/20	Wed 04/11/20	3.230,00 €
Alumno	80 hrs		Thu 08/10/20	Wed 04/11/20	3.200,00 €
Portatil	100 hrs		Thu 01/10/20	Wed 04/11/20	30,00 €
Busqueda y selección de bibliografía	56 hrs	7 days	Thu 05/11/20	Fri 13/11/20	1.128,40 €
Alumno	28 hrs		Thu 05/11/20	Fri 13/11/20	1.120,00 €
Portatil	28 hrs		Thu 05/11/20	Fri 13/11/20	8,40 €
Lectura bibliografía	96 hrs	12 days	Mon 16/11/20	Tue 01/12/20	1.934,40 €
Alumno	48 hrs		Mon 16/11/20	Tue 01/12/20	1.920,00 €
Portatil	48 hrs		Mon 16/11/20	Tue 01/12/20	14,40 €
Lectura documentación, DL Toolbox Matlab	64 hrs	8 days	Tue 19/01/21	Thu 28/01/21	1.289,60 €
Alumno	32 hrs		Tue 19/01/21	Thu 28/01/21	1.280,00 €
Portatil	32 hrs		Tue 19/01/21	Thu 28/01/21	9,60 €
Ampliación de bibliografía y hallazgo del algoritmo de Levenberg-Marquardt	200 hrs	25 days	Fri 12/03/21	Thu 15/04/21	4.030,00 €
Alumno	100 hrs		Fri 12/03/21	Thu 15/04/21	4.000,00 €
Portatil	100 hrs		Fri 12/03/21	Thu 15/04/21	30,00 €
Documentación	**708 hrs**	**161 days**	**Wed 02/12/20**	**Wed 14/07/21**	**23.080,40 €**
Redacción parte teórica de la memoria	248 hrs	34 days	Wed 02/12/20	Mon 18/01/21	10.820,40 €
Alumno	60 hrs		Wed 02/12/20	Wed 13/01/21	2.400,00 €
Tutor	120 hrs		Wed 02/12/20	Mon 18/01/21	8.400,00 €
Portatil	68 hrs		Wed 02/12/20	Fri 15/01/21	20,40 €
Inclusión de modelos y resultados en la memoria	160 hrs	20 days	Thu 06/05/21	Wed 02/06/21	3.224,00 €
Alumno	80 hrs		Thu 06/05/21	Wed 02/06/21	3.200,00 €
Portatil	80 hrs		Thu 06/05/21	Wed 02/06/21	24,00 €
Revisión de la memoria	180 hrs	15 days	Thu 03/06/21	Wed 23/06/21	6.618,00 €
Alumno	60 hrs		Thu 03/06/21	Wed 23/06/21	2.400,00 €
Tutor	60 hrs		Thu 03/06/21	Wed 23/06/21	4.200,00 €
Portatil	60 hrs		Thu 03/06/21	Wed 23/06/21	18,00 €
Redacción resumen e introducción	120 hrs	15 days	Thu 24/06/21	Wed 14/07/21	2.418,00 €
Alumno	60 hrs		Thu 24/06/21	Wed 14/07/21	2.400,00 €
Portatil	60 hrs		Thu 24/06/21	Wed 14/07/21	18,00 €
Desarrollo de modelos	**440 hrs**	**102 days**	**Fri 11/12/20**	**Mon 03/05/21**	**10.463,60 €**
Desarrollo primeros modelos con la librería PMTK	80 hrs	10 days	Fri 11/12/20	Thu 24/12/20	2.412,00 €
Alumno	40 hrs		Fri 11/12/20	Thu 24/12/20	1.600,00 €
Portatil	40 hrs		Fri 11/12/20	Thu 24/12/20	12,00 €
Licencia Matlab Anual	1		Fri 11/12/20	Thu 24/12/20	800,00 €
Definición de los problemas y entradas y salidas de los modelos	48 hrs	10 days	Fri 25/12/20	Fri 15/01/21	1.764,80 €
Alumno	16 hrs		Fri 08/01/21	Fri 15/01/21	640,00 €
Tutor	16 hrs		Fri 25/12/20	Wed 30/12/20	1.120,00 €
Portatil	16 hrs		Wed 06/01/21	Mon 11/01/21	4,80 €
Programación scripts para la generación de datasets	120 hrs	16 days	Fri 29/01/21	Fri 19/02/21	2.418,00 €
Alumno	60 hrs		Fri 29/01/21	Fri 19/02/21	2.400,00 €
Portatil	60 hrs		Fri 29/01/21	Thu 18/02/21	18,00 €
Entrenamiento de modelos y generación de resultados I	96 hrs	12 days	Mon 22/02/21	Tue 09/03/21	1.934,40 €
Alumno	48 hrs		Mon 22/02/21	Tue 09/03/21	1.920,00 €
Portatil	48 hrs		Mon 22/02/21	Tue 09/03/21	14,40 €
Entrenamiento de modelos y generación de resultados II	96 hrs	12 days	Fri 16/04/21	Mon 03/05/21	1.934,40 €
Alumno	48 hrs		Fri 16/04/21	Mon 03/05/21	1.920,00 €
Portatil	48 hrs		Fri 16/04/21	Mon 03/05/21	14,40 €
Análisis de resultados	**48 hrs**	**41 days**	**Wed 10/03/21**	**Wed 05/05/21**	**1.764,80 €**
Análisis de resultados I	24 hrs	2 days	Wed 10/03/21	Thu 11/03/21	882,40 €
Alumno	8 hrs		Wed 10/03/21	Thu 11/03/21	320,00 €
Tutor	8 hrs		Wed 10/03/21	Thu 11/03/21	560,00 €
Portatil	8 hrs		Wed 10/03/21	Thu 11/03/21	2,40 €
Análisis de resultados II	24 hrs	2 days	Tue 04/05/21	Wed 05/05/21	882,40 €
Alumno	8 hrs		Tue 04/05/21	Wed 05/05/21	320,00 €
Tutor	8 hrs		Tue 04/05/21	Wed 05/05/21	560,00 €
Portatil	8 hrs		Tue 04/05/21	Wed 05/05/21	2,40 €
Entrega TFG	**0 hrs**	**0 days**	**Mon 06/09/21**	**Mon 06/09/21**	**0,00 €**
Defensa TFG	**228 hrs**	**20 days**	**Mon 06/09/21**	**Fri 01/10/21**	**8.344,00 €**
Preparación de la presentación	228 hrs	20 days	Mon 06/09/21	Fri 01/10/21	8.344,00 €
Alumno	68 hrs		Mon 06/09/21	Fri 01/10/21	2.720,00 €
Tutor	80 hrs		Mon 06/09/21	Fri 01/10/21	5.600,00 €
Portatil	80 hrs		Mon 06/09/21	Fri 01/10/21	24,00 €
Defensa TFG	0 hrs	0 days	Fri 01/10/21	Fri 01/10/21	0,00 €
Total					**60.689,20 €**

Tabla B.1: Presupuesto

Índice de figuras

1.2.1. Arquitectura de la red neuronal de AlphaGo, toma como entrada una matriz que simboliza el tablero y consiste en 13 capas convolucionales, 12 funciones ReLu y una capa softmax. Ref: Li, Zhen-Ni —& Zhu, Can & Yu-Liang, Gao & Wang, Ze-Kun & Wang, Jiao. (2020). AlphaGo Policy Network: A DCNN Accelerator on FPGA. IEEE Access. PP. 1-1. 10.1109/ACCESS.2020.3023739. 3
1.3.1. Búsquedas al año en Google. Ref: Google Search Statistics - Internet Live Stats. (s. f.). Internet Live Stats. Recuperado 4 de septiembre de 2021, de https://www.internetlivestats.com/google-search-statistics/ 4

2.2.1. Función ReLU 8
2.2.2. Tangente hiperbólica 9
2.2.3. Función sigmoide 9
2.3.1. Esquema de una red neuronal 10
2.9.1. Aproximación de una función arbitraria por una red neuronal. Extraída de 'Neural networks and deep learning' [17] 17

4.1.1. Modelización del campo de presiones generado durante la pisada mediante funciones gaussianas. 29
4.1.2. Aproximación polinómica de la curva $f(x) = \pi x^2$ con un conjunto de 100 datos con un error aleatorio máximo del 50 %. Los crículos azules representan los datos ; la línea discontinua naranja, la función real, $f(x)$; finalmente, en amarillo continuo se representan las distintas aproximaciones 30
4.2.1. Gráficas de resultados viga empotrada 33
4.2.2. Gráficas de resultados viga empotrada 33
4.2.3. Gráficas de resultados viga empotrada 34
4.2.4. Gráficas de resultados viga empotrada 34
4.2.5. Gráficas de resultados viga empotrada 34
4.2.6. Gráficas de resultados viga empotrada 35
4.2.7. Gráficas de resultados viga empotrada 35
4.2.8. Gráficas de resultados viga empotrada 36
4.2.9. Gráficas de resultados viga empotrada 37
4.2.10. Gráficas de resultados viga empotrada 37
4.2.11. Gráficas de resultados viga empotrada 37
4.2.12. Gráficas de resultados viga empotrada 38
4.2.13. Gráficas de resultados viga empotrada 38
4.2.14. Gráficas de resultados viga empotrada 38

4.3.1. Gráficas de resultados viga simplemente apoyada 40
4.3.2. Gráficas de resultados viga simplemente apoyada 40
4.3.3. Gráficas de resultados viga simplemente apoyada 41

5.1.1. Gráficas de resultados viga empotrada 43
5.1.2. Gráficas de resultados viga empotrada 44
5.1.3. Gráficas de resultados viga empotrada 44
5.1.4. Gráficas de resultados viga empotrada 44
5.1.5. Gráficas de resultados viga empotrada 45
5.1.6. Gráficas de resultados viga empotrada 45
5.1.7. Gráficas de resultados viga empotrada 45
5.1.8. Gráficas de resultados viga empotrada 47
5.1.9. Gráficas de resultados viga empotrada 47
5.1.10. Gráficas de resultados viga empotrada 47
5.1.11. Gráficas de resultados viga empotrada 48
5.1.12. Gráficas de resultados viga empotrada 48
5.1.13. Gráficas de resultados viga empotrada 48
5.1.14. Gráficas de resultados viga empotrada 49
5.2.1. Gráficas de resultados viga simplemente apoyada 50
5.2.2. Gráficas de resultados viga simplemente apoyada 50
5.2.3. Gráficas de resultados viga simplemente apoyada 51
5.2.4. Gráficas de resultados viga simplemente apoyada 51
5.2.5. Gráficas de resultados viga simplemente apoyada 52
5.2.6. Gráficas de resultados viga simplemente apoyada 53
5.2.7. Gráficas de resultados viga simplemente apoyada 53
5.2.8. Gráficas de resultados viga simplemente apoyada 53
5.3.1. Deformadas obtenidas por los tres métodos para la viga empotrada sometida a esfuerzos constantes, las deformadas obtenidas con la red por Levenberg-Marquardt y por smmatlab se solapan 55

6.0.1. Gráficas de resultados problema de varias cargas 57
6.0.2. Gráficas de resultados problema de varias cargas 57
6.0.3. Gráficas de resultados problema de varias cargas 58
6.0.4. Gráficas de resultados problema de varias cargas 58
6.0.5. Gráficas de resultados problema de varias cargas 58

7.3.1. Evolución del número de entradas de la red necesario para describir la estructura genérica en función de los nodos de ésta. En el gráfico de la izquierda se representan cuatro rectas de evolución, que representan el número de inputs debidos a las coordenadas de los nodos, la definición de las vigas, apoyos y fuerzas. En la gráfica central solo se representan aquellos inputs aportados por la definición de las conexiones. En la último gráfica se tiene la representación de la suma de todos los inputs anteriores. 64

Índice de tablas

3.1. Muestra del dataset de viga con esfuerzos constantes 25
3.2. Muestra del dataset de viga biapoyada 27

4.1. Errores absolutos en las predicciones de la red para el problema de la viga con esfuerzos constantes usando aproximación polinómica en el dataset sin modificar 36
4.2. Errores absolutos en las predicciones de la red para el problema de la viga con esfuerzos constantes usando aproximación polinómica en el dataset modificado 39
4.3. Errores absolutos en las predicciones de la red para el problema de la viga biapoyada usando aproximación polinómica 41

5.1. Errores absolutos en las predicciones de la red para el problema de la viga con esfuerzos constantes usando el descenso del gradiente con momento 46
5.2. Errores absolutos en las predicciones de la red para el problema de la viga con esfuerzos constantes usando el algoritmo de Levenberg-Marquardt 49
5.3. Errores absolutos en las predicciones de la red para el problema de la viga biapoyada usando el descenso del gradiente con momento 51
5.4. Errores absolutos en las predicciones de la red para el problema de la viga biapoyada usando el algoritmo de Levenberg-Marquardt 54
5.5. Definición de los problemas de ejemplo 54
5.6. Soluciones ejemplo problema de viga con esfuerzos constantes 54
5.7. Soluciones ejemplo problema de viga simplemente apoyada 55

6.1. Errores absolutos para el problema de varias cargas en el conjunto de test . . 59
6.2. Datos de entrada de los 6 casos de carga 59
6.3. Comparación de las soluciones de los 6 casos dadas por la red neuronal (RN) y por la resolución convencional (EB) 59

B.1. Presupuesto 80

Bibliografía

[1] Murphy, K. P. (2012). Machine Learning. Amsterdam University Press.

[2] Schmidt, M and Lipson, H. (2009). Distilling free-form natural laws from experimental data. Science.

[3] Chang, DE. (2002). Controlled lagrangian and hamiltonian systems. PhD, California Institute of Technology.

[4] Hills, DJ. (2015). An algorithm for discovering lagrangians automatically from data. PeerJ Computer Science

[5] Ahmadi, M, Topcu, U and Rowley, C. (2018). Control-Oriented Learning of Lagrangian and Hamiltonian Systems. 2018 Annual American Control Conference

[6] Kirchdoerfer, T and Ortiz, M. (2016). Data-driven computational mechanics. Computer Methods in Applied Mechanics and Engineering

[7] Eggersmann, R., Kirchdoerfer, T., Reese, S., Stainier, L. Ortiz, M. (2019). Model-free data driven inelasticity.Computer Methods in Applied Mechanics and Engineering

[8] Hernández, Q, Badías, A, González, D, Chinesta, F and Cueto, E. (2021). Structure-preserving neural networks. Journal of Computational Physics

[9] Dot CSV. (2017, 1 noviembre). ¿Qué es el Machine Learning?¿Y Deep Learning? Un mapa conceptual — DotCSV [Vídeo]. YouTube. https://www.youtube.com/watch?v=KytW151dpqU

[10] Osiński, B., & Budek, K. (2021, 5 enero). What is reinforcement learning? The complete guide. Deepsense. Ai. https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/

[11] Kohs, G. (Director). (2017). AlphaGo [Documental]. Moxie Pictures.

[12] Google Search Statistics - Internet Live Stats. (s. f.). Internet Live Stats. Recuperado 4 de septiembre de 2021, de https://www.internetlivestats.com/google-search-statistics/

[13] Baraniuk, R., Donoho, D., & Gavish, M. (2020). The science of deep learning. Proceedings of the National Academy of Sciences, 117(48), 30029–30032. https://doi.org/10.1073/pnas.2020596117

[14] GitHub Copilot. https://copilot.github.com/

[15] Dot CSV. (2018, 19 marzo). ¿Qué es una Red Neuronal? Parte 1 : La Neurona — DotCSV [Vídeo]. YouTube. https://www.youtube.com/watch?v=MRIv2IwFTPg

[16] Sharma, A., V. (2020, 10 junio). Understanding Activation Functions in Neural Networks. Medium. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[17] Michael A. Nielsen, "Neural Networks and Deep Learning", Determination Press, 2015

[18] The MathWorks, Inc.. Options for training deep learning neural network - MATLAB trainingOptions. Matlab documentation. https://es.mathworks.com/help/deeplearning/ref/trainingoptions.html

[19] Levenberg-Marquardt backpropagation - MATLAB trainlm - MathWorks España. The MathWorks, Inc. https://es.mathworks.com/help/deeplearning/ref/trainlm.html

[20] Xue Ying 2019 J. Phys.: Conf. Ser. 1168 022022

[21] Fortuner, B. (2017, 7 marzo). Can neural networks solve any problem? Towards Data Science. https://towardsdatascience.com/can-neural-networks-really-learn-any-function-65e106617fc6

[22] Goodfellow, S. R. S. I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. https://www.deeplearningbook.org/

[23] The MathWorks, Inc.. Deep Learning Toolbox Documentation - MathWorks España. https://es.mathworks.com/help/deeplearning/index.html?s_tid=CRUX_lftnav

[24] Sarkar, Kaustav. (2009). Modeling of section forces in a continuous beam using artificial neural network. Journal of Structural Engineering (Madras).