Redes Neuronales, ¿Dónde está la magia?

Natalia Regalado

Director: Andrés Farall

Tesis de Licenciatura en Ciencias Matemáticas

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Febrero 2020

Agradecimientos

Fueron muchas las personas que me acompañaron a lo largo del camino para llegar hasta acá, y quiero agradecerle a todas y cada una de ellas.

Inicialmente quiero agradecerle a mi director Andrés, por haberme guiado en esta Tesis, gran profesor y gran profesional, aprendí mucho de él.

También quiero agradecer a todos los docentes que fueron parte de mi formación. Comenzando por Graciela, mi profesora de matemáticas del secundario, porque con su vocación y su pasión por los números supo despertar en mí el interés por este mundo.

A Itatí y Pablo, mis docentes en el CBC, por acompañarme en esa difícil etapa de transición.

A todos los docentes que tuve en Exactas, principalmente a: Gabriel Acosta, Juan Pablo Pinasco, Willy Durán, Julián Bonder, Agustin Gravano y Susana Sombielle, por marcar de alguna forma mi camino.

Al gran Marcelo Valdettaro, por sus magníficas clases de Cálculo Avanzado y Análisis Complejo, y por sus palabra de aliento.

Al jurado de esta Tesis, Pablo Groisman y Mariela Sued, por haberse tomado el tiempo de leerla, admirables docentes cuyas clases tuve el agrado de presenciar también.

A los compañeros y amigos que me dio esta carrera. A Nahuel y a Paula, por compartir conmigo las mañanas de Análisis I en ese terrorífico primer cuatrimestre de facultad.

A Malcolm, por las charlas interminables en momentos de crisis facultativa, por haber superado juntos ese intenso cuatrimestre de Cálculo Avanzado. Por ese abrazo que llegó en el momento justo.

A la única e inigualable Banda de Aplicada: Guty, Nacho, Franco, Nico e Ivan, por bancarme todas las mañanas, por el compañerismo y principalmente la buena onda, fueron años muy divertidos al lado de ustedes.

A mi gran compañero de Trabajos Prácticos y amigo Joaco, por todos sus códigos compartidos, por las tardes de ejercicios, de siestas y de charlas. Un buen amigo que me dejó esta carrera.

A mis amigas de toda la vida: Gise, Aye, Mari, Lari y Flor, por intentar mil veces entender qué significaba ser una matemática y de qué iba a trabajar cuando termine y por darme su apoyo siempre a pesar de nunca haberlo entendido completamente del todo.

A mi Tedita, única en el mundo, por esperarme todos los días con un plato rico de comida cuando llegaba de la escuela y más adelante de cursar, por todo el cariño que siempre me dió.

A mis hermanos, Lucas y Juan, por haber festejado tanto los '2' como los '10', por enseñarme que tropezar también es parte del camino.

A mi abuela, por prender una vela cada vez que rendía un examen (sí, fueron un montón de velas), y por la llamada al final del día preguntándome cómo me había ido.

A mis abuelos que arrancaron esta etapa conmigo y no pudieron llegar al final, porque desde donde sea que estén sé que siempre están conmigo.

A mi compañero de vida, Lea, me conoció en medio de esta locura y me acompañó hasta el final. Por su amor y su paciencia. Por haber estado siempre ahí.

Por último quiero agradecer a dos personas muy especiales, a mama y a papa, porque nada de esto hubiera sido posible sin el apoyo de ellos. Por haber luchado toda su vida por darme lo mejor. Por haberse esforzado para que yo pudiera dedicarme exclusivamente a la carrera. Por enseñarme con su ejemplo lo que realmente importa: ser respetuoso, honesto y responsable, y nunca bajar los brazos. Por ayudarme a cumplir mis sueños.

Esta Tesis va dedicada a ustedes dos.

Índice general

Capítulo 1

Introducción

En las últimas décadas las Redes Neuronales han atraído una gran cantidad de atención debido al poder que tienen a la hora de resolver problemas provenientes de distintos campos de investigación y de aplicación. Sin embargo, estos algoritmos no son para nada nuevos. La primera neurona artificial fue creada en 1943 por el neuropsicólogo Warren McCulloch y el lógico Walter Pits. Durante la década del 60', los investigadores se alejaron de las Redes Neuronales y se concentraron en la parte simbólica de la Inteligencia Artificial (IA). Recién hacia la década del 80' los científicos vieron el alcance real de las Redes. Hoy en día se puede decir que las Redes Neuronales son los algoritmos más potentes en el área de la IA.

En algunos campos, una Red Neuronal entrenada correctamente se puede considerar un 'experto' en la categoría de información que se le ha dado a analizar. Es un sistema tremendamente poderoso que tiene la habilidad de 'captar' cualquier fenómeno sin importar las características del mismo. Imágenes han sido clasificadas con exactitud humana Krizhevsky et al. (2012), juegos como el Go han sido 'aprendidos' por estructuras complejas Silver et al. (2016). Se han resuelto con ayuda de las redes problemas de reconocimiento de voz también Hinton et al. (2012).

Sin embargo, la razón de este éxito no se comprende completamente. Cada vez se entrenan modelos más y más complicados con resultados exitosos pero a pesar de todos sus éxitos prácticos, recién en los últimos tiempos se está formando una teoría robusta de por qué funcionan tan bien. No es para nada menor, además, el hecho de que estos algoritmos optimizan funciones no-convexas. Diferentes líneas de investigación trataron de entender el mecanismo de las Redes Neuronales complejas desde distintos ángulos. Por ejemplo, algunos trabajos tratan de entender el poder expresivo que tiene una Red dependiendo de su arquitectura, incluyendo el ancho de cada capa y la profundidad de la Red.

Existe una visión difundida que el éxito de las Redes Neuronales se basa en dos características fundamentales. Por un lado su gran flexibilidad para modelar relaciones complejas entre las variables. Por el otro, su eficiencia en la minimización de la función de pérdida, que garantiza un buen ajuste a los datos. En términos generales, en esta tesis vamos a recopilar varios trabajos realizados en el tema y tratar de responder una pregunta para nada sencilla que es: ¿por qué

funcionan de la forma en que funciona?. En particular, trabajando en contextos sencillos y mediante la utilización de datos simulados, trataremos de comprobar la hipótesis que el éxito de las redes neuronales podría basarse en la limitación práctica de sus características fundamentales, su flexibilidad y su capacidad optimizadora.

Capítulo 2

Redes Neuronales Artificiales

2.1. ¿Qué es una Red Neuronal?

A los fines de esta tesis, consideraremos que una Red Neuronal artificial es un modelo matemático que permite codificar una función continua $f : \mathbb{R}^p \to \mathbb{R}^q$. Una Red Neuronal se compone de varias capas, una capa de entrada, capas ocultas intermedias de neuronas y una capa de salida. Cada neurona tiene asignada una función de activación y las neuronas se conectan entre sí mediante vectores llamados pesos. Además cada capa tiene un término constante denominado comúnmente 'Bias'.

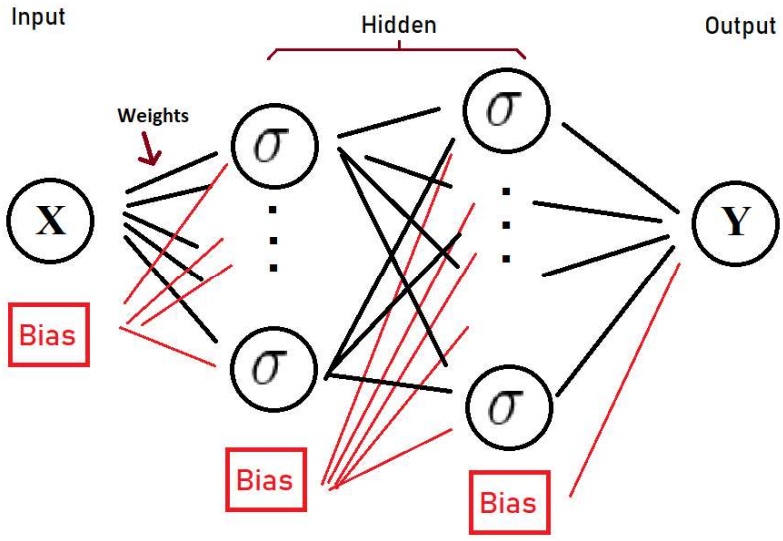

FIGURA 2.1: Arquitectura de una Red Neuronal

La cantidad de capas que tiene una red y la cantidad de neuronas que tiene cada capa es variable. Se define como profundidad (*depth*) de una Red a la cantidad de capas que tiene, incluyendo la última capa de output pero excluyendo la capa de input. Es decir una Red de profundidad 3 está formada por 2 capas ocultas. Mientras que se define la dimensión (*width*) como el máximo

de neuronas en una capa. Cuando todas las neuronas se conectan entre sí, se dice que la red es *fully connected*, como podemos ver en la figura 2.1.

La arquitectura más sencilla y la primera que existió se la conoce como Perceptrón Simple, y es una Red con una sola capa y una sola neurona en esta misma.

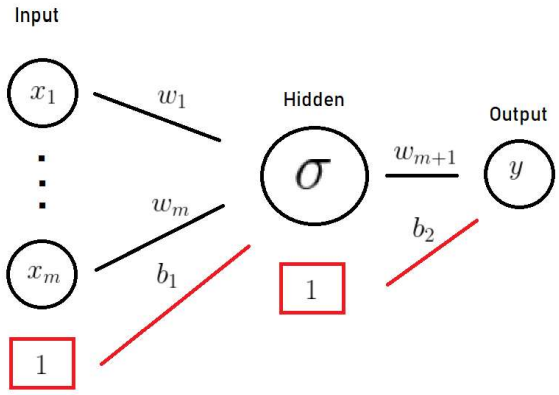

FIGURA 2.2: Perceptrón simple

Si consideramos este ejemplo, la función resultante codificada por esta arquitectura se puede escribir de la siguiente manera:

$$y = F(x_1, x_2, ..., x_m, W) = \sigma(\sum_{i=1}^{m} w_i.x_i + b_1).w_{m+1} + b_2$$

donde σ es la función de activación y $w_1, ..., w_{m+1}, b_1, b_2$ son los pesos de la Red, comúnmente llamados W. Notemos que un Perceptrón simple con m neuronas de entrada tiene m + 3 pesos, es decir W $\in \mathbb{R}^{m+3}$.

Dentro de las funciones de activación más usadas se encuentran:

- Función identidad: $f(x) = x$
- Función escalón de Heaviside : $f(x) = \begin{cases} 0 & \text{si } x < 0 \\ 1 & \text{si } x \geq 0 \end{cases}$
- Función logística: $f(x) = \frac{1}{1+e^{-x}}$
- Función tangente hiperbólica: $f(x) = tanh(x) = \frac{senh(x)}{cosh(x)}$
- Función ReLU (Rectified Linear Unit): $f(x) = max(0, x)$

Las redes son entrenadas con variables de entrada x y salida y, siendo la predicción $\hat{y} = F(x, W)$, donde W recordemos son los parámetros de la Red. Sea X entonces la muestra de entrenamiento, asumamos que tiene dimensión n, y sea $x^{(i)}$ el *i-ésimo* vector de entrenamiento, $x^{(i)} \in \mathbb{R}^m$. Luego, se busca minimizar una función de pérdida:

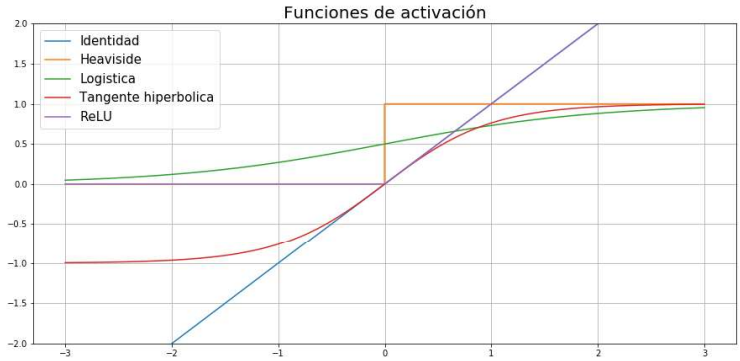

FIGURA 2.3: Funciones de Activación

$$L(W) = \frac{1}{n}\sum_{i=1}^{n} l(y^{(i)}, F(x^{(i)}, W)) \tag{2.1}$$

Para minimizar esta función la única variable es el vector de pesos W. De esta manera, entrenar una Red quiere decir hallar los valores para cada w_i, b_i de manera tal que se minimice el error.

La función de pérdida varia dependiendo el problema en cuestión. En el enfoque supervisado, los problemas se distinguen a grandes rasgos entre problemas de regresión o de clasificación. Entre las funciones de pérdidas más comunes se hallan:

1. Clasificación
   - Cross-entropy: $\frac{1}{n}\sum_{i=1}^{n}[y^{(i)}log(\hat{y}^{(i)}) + (1-y^{(i)})log(1-\hat{y}^{(i)})]$
   - Hinge: $\frac{1}{n}\sum_{i=1}^{n} max(0, 1-y^{(i)}.\hat{y}^{(i)}) \quad con\ y^{(i)} \in \{-1, 1\} \quad \forall i = 1, ..., n$
2. Regresión
   - Error cuadrático medio (MSE): $\frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y^{(i)}})^2$
   - Error absoluto medio (MAE): $\frac{1}{n}\sum_{i=1}^{n}\left|y^{(i)} - \hat{y^{(i)}}\right|$

Esta función de pérdida tienen una cantidad de parámetros que crece con la cantidad de neuronas y la cantidad de capas. Para el Perceptrón simple, es decir, una sola neurona, hay tantos parámetros como la dimensión de entrada sea, además de los bias. Si además agregamos a esa misma capa otra neurona, estos parámetros se van a duplicar. Sucesivamente si agregamos otra capa se multiplican los pesos de la red. A modo de ejemplo, una Red con 2 capas ocultas con 10 neuronas en la primera y 15 en la segunda con 5 inputs y 3 outputs tienen 273 parámetros a entrenar. Por esto, las funciones de pérdida asociadas a estas arquitecturas se hallan en espacios de grandes dimensiones. Teniendo en cuenta esto, y que además son funciones no convexas, no es para nada trivial la tarea de minimizarlas.

2.2. Gradiente Descendente

El método de minimización más utilizado para hallar el mínimo a estas funciones de pérdida se conoce como Descenso del Gradiente o Gradiente Descendente (*Gradient Descent*). El algoritmo en cuestión, es un algoritmo iterativo que se utiliza para encontrar los valores óptimos locales de una función. Para hallar el mínimo, se toman pasos proporcionales en dirección opuesta a la del gradiente (o gradiente aproximado) de la función en el punto actual. Sea $F : D \subset \mathbb{R}^m \to \mathbb{R}$ una función diferenciable en D y $x_0 \in D$, se obtiene la secuencia $x_0, x_1, x_2, ...$ tal que

$$x_{n+1} = x_n - \gamma_n \nabla F(x_n) \tag{2.2}$$

Luego, $F(x_0) \geq F(x_1) \geq F(x_2) \geq \cdots$, si x_n converge, luego converge a un mínimo local. Si F es convexa y ∇F es Lipschitz, luego el algoritmo converge a un mínimo global.

El parámetro γ_n se conoce como *learning rate* y es el paso que da el algoritmo en cada iteración. Puede depender de n o ser un valor fijo para todo el entrenamiento. Elegir este parámetro es un desafío, ya que un valor demasiado pequeño puede resultar en un largo proceso de entrenamiento que podría atascarse, mientras que un valor demasiado grande puede resultar en aprender un conjunto de pesos sub-óptimo demasiado rápido o un proceso de entrenamiento inestable.

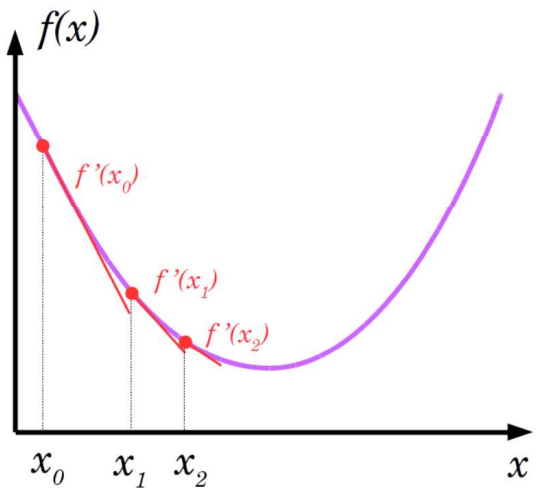

FIGURA 2.4: Descenso por Gradiente

En el caso en que F no sea convexa, como es el caso de la función de pérdida de las Redes, el Gradiente Descendente no tiene por qué alcanzar un mínimo global. Vamos a ver más adelante que numerosos autores estudiaron este problema y llegaron a importantes conclusiones.

2.2.1. Gradiente Descendente Estocástico

Recordemos que entrenar una red, quiere decir hallar el mínimo a la función de pérdida 2.1. Aplicar el método del Gradiente Descendente luego tiene un costo computacional que depende

de la dimensión de la muestra n. Al incrementar la cantidad de ejemplos de entrenamiento, se vuelve más costoso ya que, en cada iteración, se debe calcular en toda la muestra para hacer la actualización de los parámetros.

A fin de disminuir el costo computacional de este método se desarrollo el Gradiente Descendente Estocástico (SGD), en el cual los parámetros se ajustan con el gradiente de la función de pérdida de un solo ejemplo de los datos usados para entrenar el modelo. El termino estocástico hace referencia a que cada ejemplo es escogido al azar o el set es mezclado de manera aleatoria antes de comenzar el entrenamiento. En este caso el numero de pasos en el proceso de optimización, y por lo tanto la cantidad de veces que se calcula el error, es igual al número de ejemplos con los que se entrena el modelo.

El costo de usar SGD es que no necesariamente se obtienen los valores óptimos de W, pero dado que es menos exigente computacionalmente, se pueden obtener valores lo suficientemente cercanos para lograr un rendimiento adecuado del modelo.

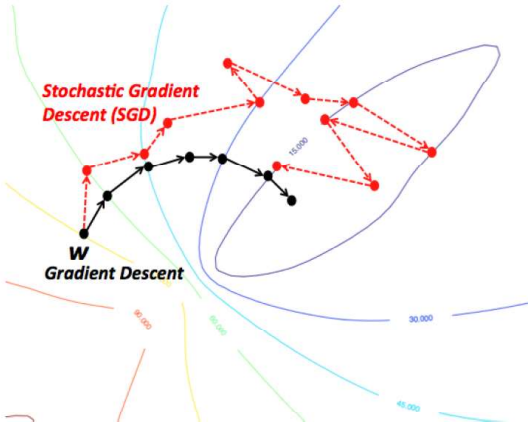

FIGURA 2.5: Descenso por Gradiente Estocástico. Fuente: https://wikidocs.net/3413

Entre el GD y el SGD se encuentra el mini-batch Gradient Descent. En este método se elige la cantidad de muestras k a utilizar en cada iteración del algoritmo. A este parámetro se lo conoce como *batch-size* y es uno de los hiperparámetros que hay que fijar al entrenar una Red. El *batch-size* se puede mover entre 1 y n, donde n es la dimensión de la muestra de entrenamiento:

- k=1 $\Rightarrow$ SGD
- k=n $\Rightarrow$ GD

Una ventaja adicional del SGD (y en general del mini batch GD), que en buena medida explica su importancia práctica, es la de reducir el potencial sobre-ajuste (overfitting). Esto se debe a que la importancia que cada observación tiene en el resultado final se ve fuertemente disminuida, ya que por azar cualquier observación participará en muy pocas iteraciones del método. De esta forma se disminuye la probabilidad que la solución final "memorice" a todas y cada una de las observaciones de la muestra.

Capítulo 3

Trabajos Previos

3.1. Aproximadores Universales

Una de las principales contribuciones en este campo proviene de algunos autores que se dedicaron a demostrar la potencia de aproximación de las Redes. Uno de los primeros desarrollos y de los más significativos fue el de Cybenko (1989). En su paper, Cybenko muestra que las sumas finitas de la forma:

$$\sum_{j=1}^{m} \alpha_j \sigma(y_j^t x + \theta_j)$$

son densas en el espacio de funciones $C([0,1]^n)$. Donde $x \in \mathbb{R}^n$, $y_j \in \mathbb{R}^n$, $\alpha_j, \theta_j \in \mathbb{R}$ y σ se refiere a cualquier función continua sigmoideal, es decir, que cumple la condición de:

$$\sigma(t) = \begin{cases} 1 & si \quad t \to +\infty \\ 0 & si \quad t \to -\infty \end{cases}$$

Es decir, que cualquier función continua del cubo unitario es aproximada arbitrariamente bien por una Red formada por una sola capa de neuronas con activación Sigmoidea. Para realizar esta demostración Cybenko utilizo el teorema de Hahn-Banach y el teorema de Representación de Riesz.

El mismo año, Hornik et al. (1989), utilizando el teorema de Stone-Weierstrass generaliza este resultado a cualquier función *squashing* como función de activación, y cualquier función medible en cualquier conjunto compacto de salida, utilizando Redes multicapa (*Deep Network*). El autor define una función *squashing* como una función $\psi : \mathbb{R} \to [0,1]$, medible y monótona creciente tal que $lim_{x \to +\infty} \psi(x) = 1$ y $lim_{x \to -\infty} \psi(x) = 0$.

Los primeros trabajos no hacen referencia ni buscan analizar la profundidad (cantidad de capas de una red) ni a la dimensión (cantidad de neuronas por capa) de la Red en cuestión, solo demuestran la existencia de una arquitectura que aproxima arbitrariamente bien a un función medible. Varios años más tarde, Lu et al. (2017) analizo la diferencia entre *Deep Networks* y

Shallow Networks (redes formadas por una sola capa) y mostraron que para cierta clase de funciones, dado un épsilon fijo, para aproximar con error menor a épsilon se necesitaba una cantidad exponencial de neuronas más en una *shallow* que en una *deep*. Para esto, trabajaron con funciones con activación ReLU y función escalón de Heaviside. En la misma linea de trabajo, Telgarsky (2015) y Telgarsky (2016), establecen un análisis en la cantidad de oscilaciones que tiene la función resultante de la Red en relación a la cantidad de capas que posee su arquitectura. Llegan así a la conclusión de que redes con muchas capas codifican funciones con muchas oscilaciones y por ende tienen mayor capacidad para aproximar funciones con muchas oscilaciones, caso contrario redes con poca cantidad de capas ocultas.

En cuanto a la dimensión de la Red, si llamamos d a la misma, Lu et al. (2017) establece que cualquier $f : \mathbb{R}^n \to \mathbb{R}$ Lebesgue integrable puede ser aproximada arbitrariamente bien por una Red *fully conected* ReLU con $d \leq n + 4$. Además muestra que si $d \leq n$ existe una clase de funciones que no puede ser aproximada. Por otro lado, Hanin (2017), demuestra que si $f : [0, 1]^n \to \mathbb{R}$ luego la cota es $d \leq n + 2$ si f es continua y $d \leq n + 1$ si f es convexa. En su siguiente trabajo, Hanin and Sellke (2017), analiza $f : [0, 1]^{n_{in}} \to \mathbb{R}^{n_{out}}$ continua, entonces d resulta estar acotado de la siguiente manera $n_{in} + 1 \leq d \leq n_{in} + n_{out}$.

3.2. Función de pérdida

Una de las principales propiedades de la superficie de pérdida es la naturaleza de sus puntos críticos, que pueden ser mínimos globales, locales o puntos silla. Varios autores centraron sus esfuerzos en poder caracterizar esto para distintas Redes Neuronales.

La clase más simple de estas son las lineales, es decir, aquellas cuya función de activación es precisamente lineal. Para ellas el primer trabajo fue de Baldi and Hornik (1989). En este trabajo, sus autores dieron una forma analítica de los puntos críticos para una Red Neuronal lineal con una capa oculta y función de pérdida cuadrática. Baldi and Lu (2012) también estudiaron los autoencoder lineales con una capa oculta y mostraron la equivalencia entre los mínimos locales y mínimos globales. Más recientemente, otros autores estudiaron las Redes lineales:Kawaguchi (2016), Lu and Kawaguchi (2017),Yun et al. (2017), en sus trabajos establecieron la equivalencia entre mínimos locales y mínimos globales bajo distintos supuestos. Particularmente, Yun et al. (2017) estableció una condición suficiente y necesaria para que un punto crítico de una Red lineal sea un mínimo global. Un resultado similar fue enunciado en el trabajo de Freeman and Bruna (2016) bajo la condición de que las capas intermedias tengan mayor dimensión que las de entrada y salida.

Mientras que el objetivo sigue siendo para nada trivial ya que se trata con un problema de optimización no convexo, las redes lineales no son la clase más interesante para considerar ya que en el fondo solo son eficientes a la hora de codificar funciones lineales. Por su parte, otro grupo de autores se dedicaron a estudiar Redes no lineales. En este campo, uno de los primeros trabajos fue realizado por Yu and Chen (1995). En él, los autores estudiaron Redes no lineales

con una capa oculta y función de activación Sigmoidea y demostraron que cada mínimo local es un mínimo global asumiendo que el número de neuronas en la capa de entrada sea igual al número de muestras.

Gori and Tesi (1992) consideraron Redes multicapa con estructura piramidal y mostraron que los puntos críticos de una columna de rango completo alcanzan una pérdida de cero cuando el tamaño de muestra es menor que la cantidad de neuronas en la capa de entrada. Este resultado fue generalizado más adelante a una clase mayor de Redes no lineales por Nguyen and Hein (2017), donde además mostraron que los puntos críticos que cumplen que el determinante de su matriz Hessiana es distinto de cero son mínimos globales. Por otra parte, a la hora de caracterizar la superficie de pérdida para redes no lineales también, un interesante enfoque fue realizado por Choromanska et al. (2015). Aleatorizando la parte no-lineal de una Red con activación ReLU y agregando algunos supuestos adicionales, ellos relacionan el modelo con el modelo de Spin-glass Hamiltoniano. En este modelo, el valor del mínimo local esta cerca al del mínimo global y la cantidad de mínimos locales 'malos' decrece rápidamente con la distancia al mínimo global. Luego, Kawaguchi (2016) elimino los supuestos de Choromanska et al. (2015) y estableció la equivalencia entre mínimos locales y mínimos globales reduciendo la función de pérdida de la Redes no lineales a la de las Redes lineales. Soudry and Carmon (2016) mostraron que una Red no lineal de dos capas ocultas no tiene mínimo local 'mal' diferenciable. Feizi et al. (2017) estudió Redes con una capa oculta con parámetros restringidos en un conjunto de direcciones y demostró que casi todo mínimo local resultaba ser un mínimo global.

Uno de los últimos avances en la literatura, Zhou and Liang (2017) engloba a los anteriores y muestra que:

- Para la función de pérdida cuadrática en una Red lineal con una capa oculta el autor caracteriza con condiciones necesarias y suficientes la forma analítica de los puntos críticos y los mínimos globales. En este trabajo, los autores generalizan el estudio de Hornik et al. (1989) y puntualmente, muestran que cualquier mínimo local es un mínimo global y que cualquier otro punto crítico es un punto silla, bajo ningún supuesto sobre la dimensión de los parámetros de la Red ni de los datos de entrada.

- Para la función de pérdida cuadrática en una *deep* lineal establecen una caracterización completa de la forma analítica de los puntos críticos y los mínimos globales, generalizando el trabajo de Kawaguchi (2016) bajo ningún supuesto en la dimensión de los parámetros de la Red ni de los datos de entrada.

- Para la función de pérdida cuadrática en una Red con una capa oculta y activación ReLU, proveen una caracterización completa de existencia y forma analítica de los puntos críticos con parámetros de entrada en ciertos tipos de regiones. Particularmente, en el caso en que hay una única neurona en la capa oculta, los resultados se generalizan a parámetros de entrada en cualquier espacio.

3.3. Poder expresivo

Ahora bien, habiendo demostrado varios autores que las Redes alcanzan perdida 0 bajo diferentes supuestos y para distintas arquitecturas la pregunta que surge es:

¿Cómo y por qué es que estos modelos generalizan tan bien?

Es decir, como es que la diferencia entre el error de la muestra de entrenamiento y la muestra de testeo es tan chico (lo que las termina convirtiendo en modelos predictivos exitosos). Varios autores analizaron este problema, entre ellos, se destaca el trabajo de Zhang et al. (2016) donde la pregunta inicial que se hacen los autores es: ¿Qué es lo que distingue una Red Neuronal que generaliza bien de una que no?. Específicamente los autores se centraron en analizar problemas de clasificación de imágenes entrenando Redes Convolucionales utilizando como método de optimización el SGD y mediante numerosos experimentos, mostraron que estas arquitecturas se ajustan fácilmente a etiquetas definidas de forma aleatoria a los datos de entrenamiento.

Por su parte, para caracterizar la expresividad de las redes neuronales, y comprender cómo la expresividad varía con los parámetros de la arquitectura, en su artículo Raghu et al. (2017) los autores presentan un conjunto interrelacionado de medidas de expresividad y muestran estrechos límites exponenciales en el crecimiento de estas medidas en la profundidad de las redes. Además, a partir de sus experimentos, concluyen que en la práctica las redes pueden ser más sensibles a pequeñas perturbaciones en los pesos en las capas inferiores. También desarrollan un nuevo método de regularización: la regularización de la trayectoria.

Por otro lado, resaltamos el trabajo Li et al. (2016), en él los autores buscan responder la siguiente pregunta: ¿Las redes aprenden conjuntos de características radicalmente diferentes que tienen un rendimiento similar, o exhiben lo que ellos llaman 'aprendizaje convergente', es decir, sus representaciones de características aprendidas son en gran medida lo mismo?. Para encarar el problema se centran en el caso particular de arquitecturas complejas de Redes Convolucionales utilizadas en clasificación de imágenes. Los autores definen un método para cuantificar la similitud de características entre diferentes Redes neuronales y luego de varios experimentos concluyen que algunas características se aprenden repetidamente en múltiples redes, pero otras características raras no siempre se aprenden.

3.4. Gradiente Descendente

A la hora de optimizar la función de pérdida el método utilizado comúnmente es el método del Gradiente Descendente. Para el mismo se han realizado varios trabajos, el más reciente, Du et al. (2018) muestra que el método del gradiente alcanza el cero en el error de entrenamiento en tiempo polinomial para una Red sobre-parametrizada del tipo ResNet. Algunos años atrás, Lee et al. (2016), demostraron que si una función $f : \mathbb{R}^D \to \mathbb{R}$ es C^2 y satisface que sus puntos críticos son o bien un mínimo global o un punto 'strict saddle' (ie $\bigtriangledown^2 f$ tiene al menos un autovalor menor estricto a 0) luego el método del Gradiente con un inicialización random y un

paso constante lo suficientemente chico converge a un mínimo local o $-\infty$ casi seguramente. A su vez, en el trabajo de Zou et al. (2018), sus autores estudiaron el problema de clasificación binaria para Redes sobre-parametrizadas con activación ReLU y una familia de funciones de pérdida y mostraron que con puntos de inicialización adecuados para los métodos, tanto el Gradiente Descendente como el Gradiente Descendente Estocástico, pueden hallar el mínimo global bajo leves supuestos de los datos de entrada.

Por su parte, en el trabajo Maennel et al. (2018), sus autores estudiaron el caso de las redes con una sola capa y activación ReLU, dándole como hiperparámetro un λ (*learning rate*) infinitesimal al GD y pesos iniciales definidos como $w_i = \lambda.v_i$, donde v_i pertenece a una distribución normal, y concluyeron que independientemente de la cantidad de neuronas que tenga la red, esta converge a una de las finitas funciones "simples"que dependen solo de la muestra de entrenamiento.

Teniendo en cuenta este marco teórico que engloba a las Redes, el propósito de esta tesis es analizar cómo es que una Red sobre-parametrizada no cae en la sobre-ajuste de los datos de entrada, considerando que son arquitecturas que por lo estudiado anteriormente pueden aproximar cualquier función continua y con métodos de optimización como el Gradiente Descendente la función de pérdida alcanza su mínimo global bajo ciertas condiciones que generalmente se satisfacen.

Capítulo 4

Interpolación vs generalización

El objetivo de esta tesis es tratar de entender el funcionamiento de las Redes Neuronales en lo concerniente a la capacidad de modelado en un contexto simple de regresión. Como vimos en el capítulo 3, existen numerosos trabajos teóricos que, bajo ciertas condiciones, prueban que el mínimo absoluto existe y que mediante el Gradiente Descendente la Red lo alcanza. Sin embargo, en la práctica, muchas veces los ajustes que se obtienen con las redes son muy parsimoniosos. Lejos de sobre-ajustar a los datos (memorizarlos), la Red logra generalizar exitosamente el patrón subyacente que explica el comportamiento de la mayoría de los datos. Sorprendentemente, esta situación ocurre incluso en contextos en los cuales la Red tendría la capacidad teórica de interpolar los datos, alcanzando así un mínimo global de valor 0 para la función de pérdida.

Para analizar este problema, nos concentramos en entender un ejemplo concreto sencillo de una $f : \mathbb{R} \to \mathbb{R}$ utilizando como herramienta el software estadístico R. Generamos una muestra de la siguiente manera: definimos uniformemente N valores para x $\in (-2, 2)$ y luego evaluamos la función $y = f(x) + z$, donde $f(x) = 3 + 4x^2$ y $z \sim N(0, 1/4)$. Comenzamos definiendo N=60 y generando así, nuestra primer muestra. Utilizamos la librería *neuralnet*, un paquete de R especializado en la calibración de redes neuronales y definimos una Red Neuronal formada de una capa oculta de 100 neuronas *fully connected*. La función de activación elegida fue la función logística, la función de pérdida fue el MSE y el algoritmo de optimización fue GD. Observemos que con esta arquitectura, la Red construida tiene 301 parámetros (pesos) a entrenar, es decir que si tomamos una muestra de 60, la Red resulta sobre-parametrizada.

Realizamos el entrenamiento con diferentes combinaciones de hiperparámetros del optimizador. Movimos el *learning rate* y el *treshold* (criterio de parada). Como podemos ver en el gráfico 4.1, para todas las combinaciones de hiperparámetros evaluadas, el algoritmo no logra la interpolación de los datos.

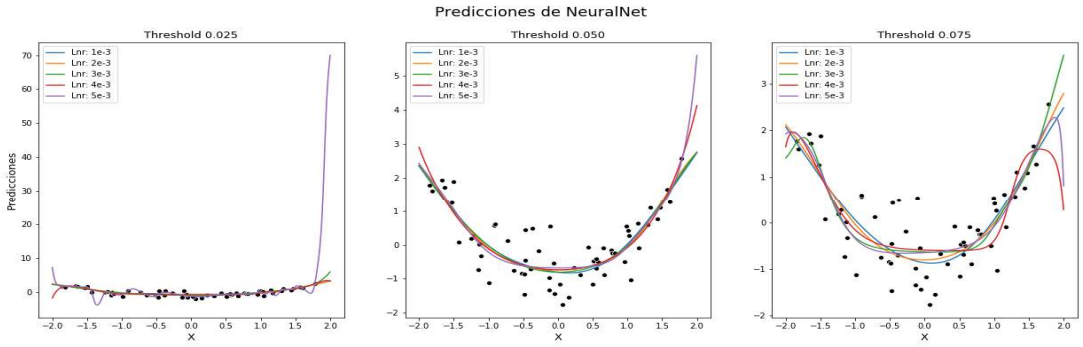

FIGURA 4.1: Predicciones obtenidas por Neural Net

Para modelar y entrenar redes más complejas pasamos a la librería *Keras*. Esta librería nos permite definir una Red con cualquier arquitectura deseada, setear parámetros de cantidad de neuronas y de cantidad de capas y la función de activación elegida para cada capa, también podemos fijar si cada capa de la Red va a ser *fully connected* o no. A la hora de compilar el modelo debemos fijar la función de pérdida, el método de optimización y la métrica con la cual queremos evaluar esa pérdida. Si elegimos SGD o GD como método de optimización, debemos fijar los parámetros de *learning rate* y *batch size*. Por último, el hiperparámetro *epochs* indica la cantidad de veces que itera el algoritmo de optimización sobre el dataset de entrenamiento.

Otra motivación para usar la librería *Keras* de R, es que la misma nos permite definir arbitrariamente los pesos iniciales con los que arranca el método GD. Esto nos será de mucha utilidad en las próximas secciones en las que modificaremos los pesos para evaluar los diferentes desempeños de la Red.

Todas las redes que siguen, entrenadas para esta tesis son *fully connected* y tienen los siguientes hiperparámetros:

1. Función de activación : ReLU
2. Función de Pérdida : Error cuadrático medio (MSE)
3. Métrica para el error : Error absoluto medio (MAE)
4. Optimizador : SGD, el hiperparámetro *batch size* es fijado siempre como la dimensión de la muestra, convirtiendo al SGD en un GD

Los pesos de las redes se van a definir inicialmente como uniformes (0,1), más adelante esto lo modificaremos. Los hiperparámetros *learning rate* y *epochs* son diferentes para cada ejemplo y serán indicados en cada caso.

Inicialmente, generamos una muestra con la misma función que antes, $f(x) = 3 + 4x^2$, más el ruido normal y con N=60, y para estos datos vamos a entrenar varias redes con diferentes hiperparámetros. Vamos a definir ahora una estructura más compleja que en el ejemplo anterior, en este caso la Red va a constar de dos capas ocultas, una primer capa de 10 neuronas, y otra

segunda de 100, dejando así 1221 parámetros a entrenar como se puede ver en la salida de la definición del modelo de Keras: 4.2.

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 10)	20
dense_2 (Dense)	(None, 100)	1100
dense_3 (Dense)	(None, 1)	101

Total params: 1,221
Trainable params: 1,221
Non-trainable params: 0

FIGURA 4.2: Summary del Modelo

Veamos que cual es el comportamiento de una red de esta arquitectura al ser entrenada con diferentes combinaciones de hiperparámetros. Para eso elegimos 4 valores diferentes de *learning rate* y 3 valores distintos de *epochs*.

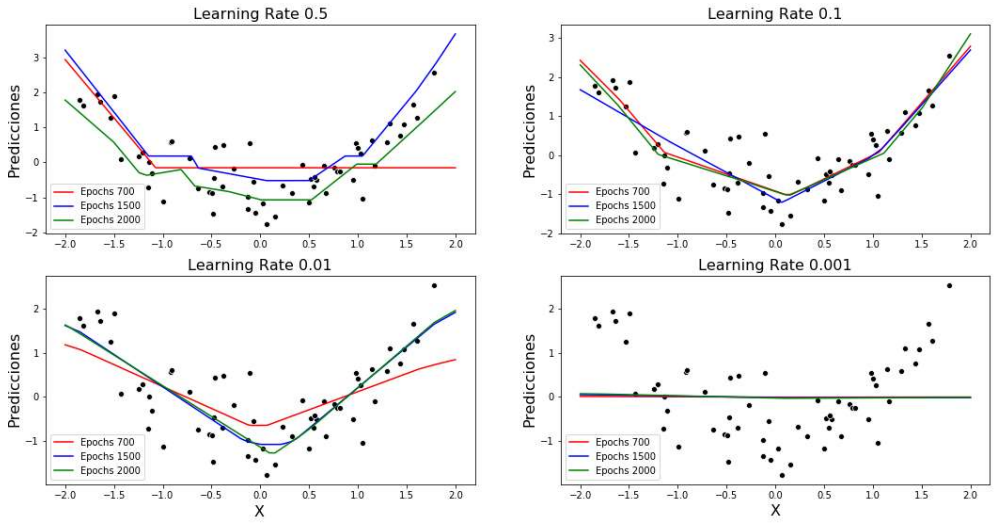

FIGURA 4.3: Comparación de los hiperparámetros del optimizador

Como podemos ver en la figura 4.3 para diferentes hiperparámetros obtenemos predicciones muy diferentes. Si tomamos un *learning rate* muy chico, como 0.001, por más *epochs* que le demos para entrenar, la Red no logra acoplarse a los datos ya que el paso que puede dar en cada iteración es muy ínfimo como para modificar significativamente el resultado. Para un *learning rate* apropiado, como 0.1 o 0.01, a mayor *epochs* mejor predicción obtenemos ya que le damos mayores iteraciones al algoritmo optimizador para acercarse al óptimo.

En la tabla 4.1 podemos ver los diferentes valores de MSE obtenidos para cada combinación de hiperparámetros.

Epochs	LR 0.5	LR 0.1	LR 0.01	LR 0.001
700	0.698	0.268	0.468	0.977
1500	0.372	0.309	0.337	0.965
2000	0.386	0.265	0.331	0.955

Tabla 4.1: Tabla de Pérdidas para diferentes hiperparámetros

Cabe destacar que ninguna combinación de hiperparámetros se acerca a la interpolación de los datos, es decir, que en ningún caso la función de pérdida alcanza la pérdida 0, al igual que en el ejemplo anterior con una arquitectura más sencilla y utilizando otra librería. Para esta arquitectura también, en todos los casos (menos para el *learning rate* 0.001) se obtienen aproximaciones parsimoniosas a la función f. En el caso del *learning rate* 0.001 lo que podemos ver que pasa es que el paso del algoritmo es demasiado chico y sin importar cuán grande sea el número de *epochs*, el algoritmo de optimización no logra moverse lo suficiente del punto de salida como para llegar a una predicción de los datos.

Para asegurarnos que la ausencia de interpolación no es fruto de una mala elección de las pocas combinaciones de los hiperparámetros *learning rate* y *epochs* que hicimos, generamos con la misma muestra y misma arquitectura definida anteriormente, 500 entrenamientos definiendo en cada uno al parámetro *epochs* uniformemente entre 500 y 3000 y al *learning rate* entre 0.005 y 0.5 respectivamente.

En la figura 4.4 podemos ver como se distribuye la pérdida de las simulaciones respecto a estos dos hiperparámetros *epochs* y *learning rate*. En ambos gráficos agregamos también la pérdida propia de los datos generada por el ruido agregado a la función f y podemos ver que en el mejor de los casos la Red alcanza ese valor. La pérdida para este rango de hiperparámetros se acumula alrededor del 0.3, mientras que para un *learning rate* mayor a 0.4 el algoritmo resulta inestable obteniendo pérdidas mayores a la que se obtendría si se ajustaran los datos con la media. El hiperparámetro más irregular resulta ser el de *epochs*, lo cual también se puede ver en la tabla 4.1, donde para una misma cantidad de iteraciones, la pérdida da muy diferente según el paso elegido.

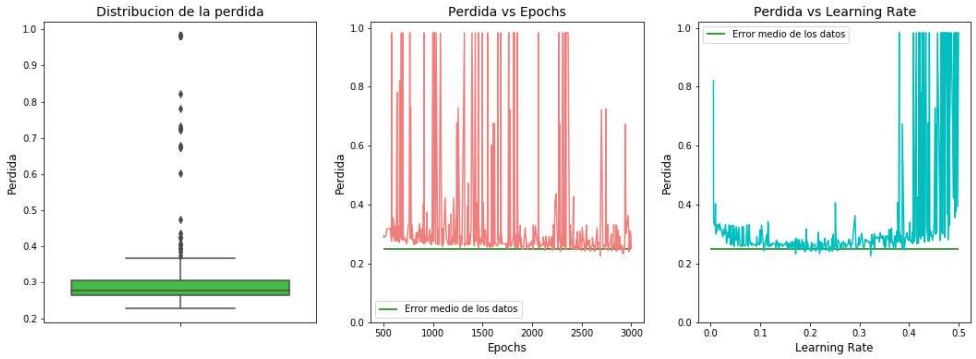

Figura 4.4: Análisis de la pérdida respecto de los hiperparámetros del optimizador

Resulta importante mencionar de nuevo que para ninguna combinación de estos parámetros la Red logra interpolar a los datos, la pérdida mínima obtenida fue de 0.22. Es interesante observar además que para la mayor parte de los valores 'razonables' de *learning rate* (valores moderados), la pérdida obtenida se aproxima mucho al valor del desvío del ruido con el que se generaron los datos.

Para lo que sigue, luego del análisis de hiperparámetros realizado, decidimos fijar como *learning rate* 0.1 y *epochs* 1500.

4.1. Diferentes tamaños de muestras

Para tratar de forzar una situación de interpolación, vamos a realizar el mismo análisis que hicimos previamente, pero con diferente tamaño N de muestra. Tomando ahora como *learning rate* 0.1 y *epochs* 1500, se entrenaron dos modelos distintos, uno con una muestra de N=60 y otro con N=5, siendo N el tamaño de la muestra para entrenar la red. En ambos casos se corrieron 100 entrenamientos de las redes con la arquitectura presentada anteriormente. Se guardaron los pesos iniciales y resultantes, la pérdida obtenida y se utilizaron las redes entrenadas para predecir luego una secuencia de valores de x entre -2 y 2. Los resultados obtenidos se pueden ver en la figura 4.5.

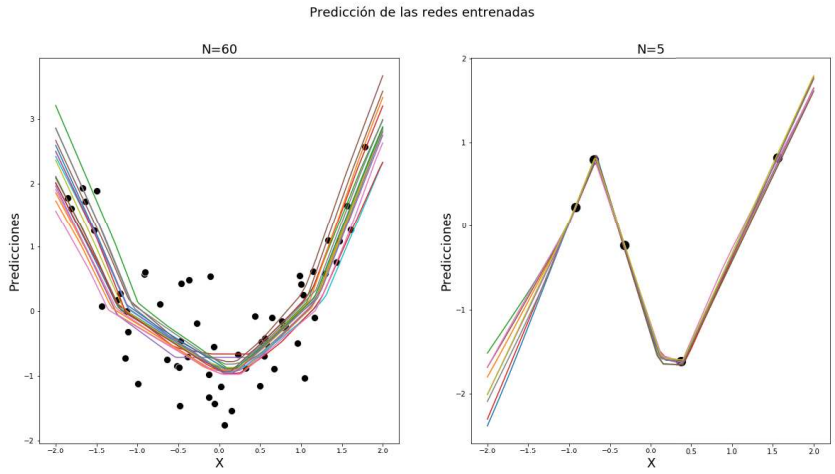

FIGURA 4.5: Resultado obtenido al entrenar la Red con 5 y 60 observaciones respectivamente

En la figura 4.5 se puede ver que tenemos comportamientos distintos para cada valor de N. Para N=60 en todas las corridas se alcanza una representación parsimoniosa de los datos, y de la función aproximada. De haber alcanzado el mínimo global, lo que veríamos sería una función que interpolaría los puntos dados. Acá es importante mencionar que el poder de la Red justamente está en hallar una codificación parsimoniosa. En caso de que la Red hubiese sobre-ajustado los datos, hubiésemos caído en un problema de *overfitting*, en cuanto no hubiera sido este un algoritmo exitoso para predecir.

Es sorprendente el hecho que entrenando una Red con 1221 parámetros, cantidad muy superior a la de los datos (60), el resultado sea completamente parsimonioso. De alguna forma, si bien las redes son aproximadores universales, como se demostró en la década del '90 y como muchos otros autores mostraron después, en la práctica siempre obtenemos funciones suaves donde no se alcanza el mínimo global.

Para N=5, el resultado cambia, la Red logra aproximar los 5 puntos interpolándolos y por ende la función de pérdida en este caso si alcanzó el mínimo global. Para lograr la interpolación en este caso se tuvo que aumentar el parámetro *epochs* de la Red a 2000 ya que para valores menores estaba cerca pero no lo conseguía.

Con el objetivo de analizar la diferencia del comportamiento vamos a ver las pérdidas obtenidas en cada caso por los entrenamientos.

Vamos ahora a comparar por un lado la pérdida resultante del entrenamiento y por el otro, la pérdida de un ajuste constante igual a la media de los N puntos, repitiendo esto para N=5 y N=60. En ambos casos se utiliza como función de pérdida el error cuadrático medio (MSE). En cada escenario (N=5 y N=60) realizamos 100 entrenamientos, con la finalidad de verificar la sistematicidad en la diferencia de comportamientos entre ambos escenarios. En la figura 4.6 podemos ver para un entrenamiento en cada caso las diferencia de estas pérdidas.

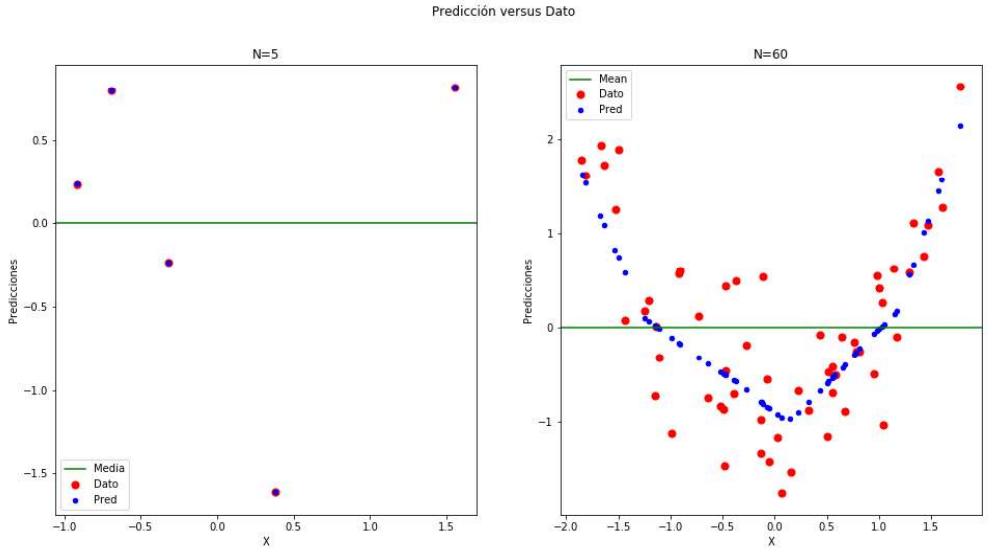

FIGURA 4.6: Predicciones versus datos para N=5 y N=60

Analíticamente, calculamos el promedio del MSE para los 100 entrenamientos en cada caso y obtuvimos los siguientes resultados:

Pérdida	$N = 5$	$N = 60$
Pérdida del ajuste de la Red	0.002387	0.258802
Pérdida del ajuste constante de los datos	0.800000	0.983333

TABLA 4.2: Tabla de Pérdidas

En la tabla 4.2 se puede ver la diferencia entre ambos N, en caso de N=5 la pérdida final del entrenamiento es casi 0, y como resultado de esto se interpolan los 5 puntos, mientras que para N=60 esto no pasa.

En el siguiente sección procedemos a analizar el comportamiento en ambos casos de las redes, considerando sus pesos de inicio y finalización (luego de ser la Red entrenada) para buscar similitudes y diferencias entre sus comportamientos.

4.2. Análisis de las trayectorias de los pesos

Como vimos anteriormente, con la arquitectura definida de la Red, obtenemos vectores de pesos que se hallan en un espacio $\mathbb{R}^{1221}$. Los pesos iniciales de cada Red y los finales, luego de ser entrenada la Red, se guardaron en una vector de pesos, obteniendo entonces dos vectores, una para N=5, donde recordamos la Red logro interpolar los datos de entrada, y otro para N=60 donde la Red alcanzó una estructura suave y parsimoniosa de los datos de entrada. Vamos a definir métricas para analizar estos dos vectores analíticamente primero y luego vamos a utilizar un algoritmo de reducción de dimensión para poder analizarlos gráficamente.

4.2.1. Métricas para los pesos

Sean $\bar{W}^b \in \mathbb{R}^{1221}$ los pesos iniciales de la Red y $\bar{W}^e \in \mathbb{R}^{1221}$ los pesos que alcanza la Red entrenada, y S la cantidad de simulaciones realizadas. Sean las matrices de distancias D_b y D_e, tales que $D_b[i,j] = \left\|\bar{W}_i^b - \bar{W}_j^b\right\|_2$, $D_e[i,j] = \left\|\bar{W}_i^e - \bar{W}_j^e\right\|_2$. Definimos las siguientes métricas:

$$B = \frac{1}{S}\sum_{j=1}^{S}\left(\frac{1}{S-1}\sum_{i=1}^{S} D_b[i,j]\right) \tag{4.1}$$

$$E = \frac{1}{S}\sum_{j=1}^{D}\left(\frac{1}{S-1}\sum_{i=1}^{S} D_e[i,j]\right) \tag{4.2}$$

$$T = \frac{1}{S}\sum_{i=1}^{S}\left\|\bar{W}_i^b - \bar{W}_i^e\right\|_2 \tag{4.3}$$

De esta forma, B representa la distancia promedio entre los vectores de pesos iniciales, que da una idea de cuan dispersos se hallan estos vectores en el espacio, mientras que E hace lo análogo para los finales. A fines prácticos en el contexto de esta tesis nos vamos a referir como trayectoria de los pesos al vector que une los pesos iniciales y los pesos finales. Bajo esta definición, T es el promedio de las normas de la trayectoria para cada entrenamiento. En este análisis S=100 ya que se realizaron 100 simulaciones para cada N. En la tabla 4.3 se encuentran los resultados obtenidos.

Métrica	$N = 5$	$N = 60$
B	2.35	2.35
E	6.15	5.72
T	3.52	3.57

TABLA 4.3: Tabla de distancias

Podemos observar que tanto para N=5 como para N=60 B vale lo mismo, y esto tiene sentido ya que en ambos casos los pesos se inicializaron con la misma distribución uniforme, es decir que la 'dispersión' de cada vector de pesos del espacio de entrada es la misma. En ambos casos, además, las otras métricas E y T también son comparables en magnitud. Es decir que las trayectorias en longitud recorridas por los pesos de ambas redes hasta llegar al mínimo de la función de pérdida se asemejan, y en ambos casos esto da como resultado vectores finales que se encuentran a mayor distancia entre sí que los iniciales, es decir, con mayor dispersión, inclusive para el caso de N=5 esta dispersión es un poco mayor. Esto resulta interesante ya que vale la pena recordar que estamos comparando un caso de interpolación de los datos (N=5) con un caso de ajuste parsimonioso de los datos (N=60), sin embargo, los pesos se comportan de manera similar en los entrenamientos.

Con el objetivo de visualizar los pesos de la Red de alguna manera, y debido a que estos viven en un espacio de dimensión 1221, recurrimos a una herramienta de reducción de dimensionalidad para poder visualizarlos en un espacio menor y poder analizarlos gráficamente.

4.2.2. t-SNE

T-Distributed Stochastic Neighbor Embedding (t-SNE) es una técnica de reducción de dimensionalidad que sirve para visualizar datasets de grandes dimensiones, presentada en el trabajo de Maaten and Hinton (2008). Esta técnica le asigna un punto en un espacio de 2 o 3 dimensiones a cada vector del espacio original de dimensión mayor. Es una variación de Stochastic Neighbor Embedding (Hinton and Roweis (2003)) que es mucho más simple de optimizar y produce significativamente mejores visualizaciones reduciendo la tendencia de acumular puntos en el centro del eje que realizaba la técnica anterior. Esta tendencia es referida como *crowding problem*.

Stochastic Neighbor Embedding (SNE) comienza convirtiendo las distancias Euclideanas entre puntos del espacio de alta dimensión en probabilidades condicionales que representan similitudes. La similitud del punto x_j con el x_i está dada por la probabilidad condicional $p_{j|i}$, tal que x_i elegiría x_j como su vecino si los vecinos fuesen elegidos en proporción a su densidad de probabilidad bajo una Gaussiana centrada en x_i. Para puntos cercanos, $p_{j|i}$ es relativamente alta, mientras que para puntos distantes, $p_{j|i}$ va a ser infinitesimal. La probabilidad condicional $p_{j|i}$, está dada por:

$$p_{j|i} = \frac{exp(-\left\|x_i - x_j\right\|^2 / 2\sigma_i^2)}{\sum_{k \neq i} exp(-\left\|x_i - x_k\right\|^2 / 2\sigma_i^2)}$$

donde σ_i es la varianza de la Gaussiana centrada en x_i. Como solo importa modelar similitudes par a par, el valor de $p_{i|i}$ se fija en 0. Para las contrapartes en el espacio de menor dimensión, y_i e y_j, se computa la misma probabilidad condicional, notada $q_{j|i}$. Se fija como varianza de la Gaussiana utilizada para computar esta probabilidad, $\frac{1}{\sqrt{2}}$. Luego, $q_{j|i}$, queda dada por:

$$q_{j|i} = \frac{exp(-\left\|y_i - y_j\right\|^2)}{\sum_{k \neq i} exp(-\left\|y_i - y_k\right\|^2)}$$

Por la misma razón de antes, se fija $q_{i|i} = 0$. Si el mapeo de puntos y_i e y_j modela correctamente la similitud entre los puntos de alta dimensión x_i y x_j entonces las probabilidades condicionales ,$p_{j|i}$ y $q_{j|i}$ van a ser iguales. Motivado por esta observación, SNE busca encontrar puntos en el espacio de menor dimensión, y_n, que minimicen la diferencia entre $p_{j|i}$ y $q_{j|i}$. Para esto, una medida natural de fidelidad con la que $q_{j|i}$ modela $p_{j|i}$ es la divergencia de Kullback-Leibler. SNE minimiza la suma de las divergencias KL sobre todos los puntos usando el método del gradiente descendente. La función de costo C a minimizar, queda dada por:

$$C = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i}\, log \frac{p_{j|i}}{q_{j|i}} \tag{4.4}$$

donde P_i representa la distribución de probabilidad condicional sobre todos los puntos dado x_i, y Q_i representa la distribución de probabilidad condicional de todos los puntos mapeados dado y_i. Hay un costo grande por usar puntos separados para representar datos que originalmente estaban cerca, es decir, por usar un valor pequeño de $q_{j|i}$ para modelar un valor grande de $p_{j|i}$, pero hay un menor costo en representar de forma cercana puntos que originalmente estaban distanciados. En otras palabras, la función de costo del SNE hace foco en retener la estructura local de los datos originales en el mapeo.

El parámetro que resta por definir es la varianza de la Gaussiana, σ_i, centrada en cada punto x_i. La realidad es que no hay un valor de σ_i que sea el óptimo para todos los puntos del dataset ya que la densidad de los puntos varia en el conjunto de datos. Cualquier valor de σ_i fijo induce una distribución de probabilidad fija, P_i fija sobre todos el conjunto de puntos. Esta distribución tiene una entropía que crece en forma proporcional con σ_i. SNE realiza una búsqueda binaria para el valor de σ_i que induce una P_i con un parámetro denominado *Perplexity* fijado por el usuario.*Perplexity* está definida como:

$$Perp(P_i) = 2^{H(P_i)}$$

donde $H(P_i)$ es la entropía de Shanon de P_i medida como $H(P_i) = -\sum_j p_{j|i}\, log_2(p_{j|i})$

El parámetro *Perplexity* puede ser interpretado como una medida suave de la efectividad del número de vecinos. Notar que el valor de *Perplexity* es monótonamente creciente respecto de la varianza σ_i. La performance del SNE cambia en forma robusta para *Perplexity*, los valores típicos se mueven entre 5 y 50.

El algoritmo aplica el Gradiente Descendente para minimizar la función de costo definida en 4.4.

Aunque SNE construye visualizaciones razonablemente buenas, el problema radica en la dificultad a la hora de optimizar la función de costo. Los autores en Maaten and Hinton (2008) además citan un problema al que llaman *crowding problem* que yace en SNE. Para resolver ambas situaciones, presentan una nueva técnica, T-Distributed Stochastic Neighbor Embedding (t-SNE), que difiere en la original SNE en dos aspectos:

1. Usa una forma simétrica de la función de costo que simplifica el Método del Gradiente Descendente.

2. Una distribución t-Student reemplaza a la Gaussiana para computar la similitud entre dos puntos del espacio de menor dimensión. t-SNE utiliza una distribución pesada en la cola en el espacio de menor dimensión para aliviar tanto el *crowding problem* como los problemas de optimización de SNE.

4.2.2.1. Resultados de t-SNE

Aplicamos el algoritmo de t-SNE incluido en librería *Scikit-learn* de *Python* con distintos parámetros de *Perplexity* a los pesos iniciales de las redes y los pesos finales, luego del entrenamiento. Obtuvimos los siguientes resultados.

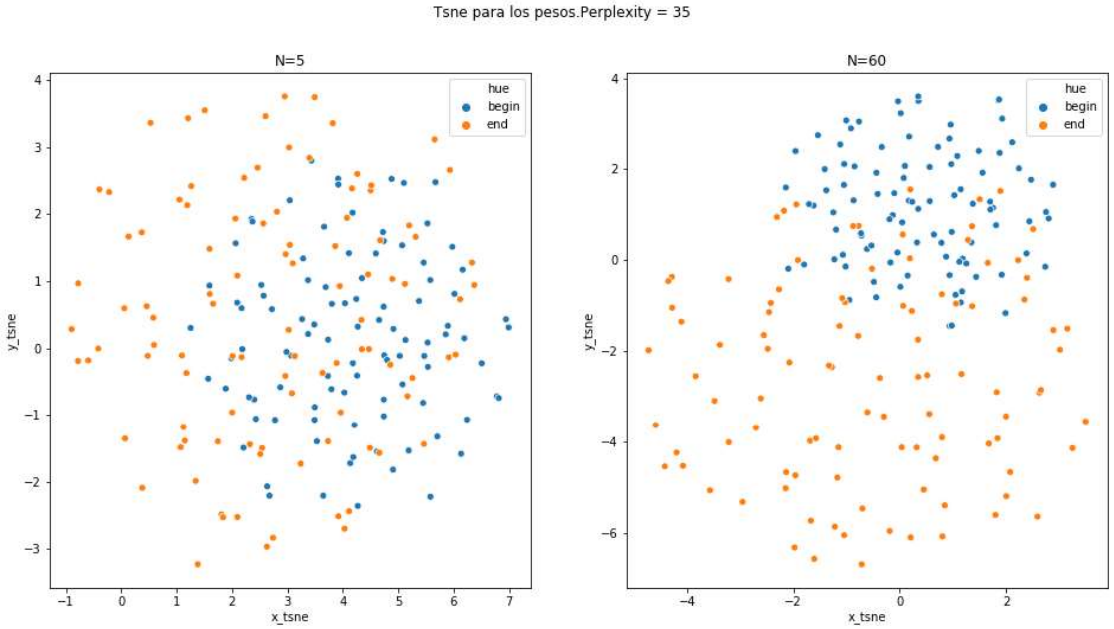

FIGURA 4.7: Resultados de t-SNE con Perplexity=35

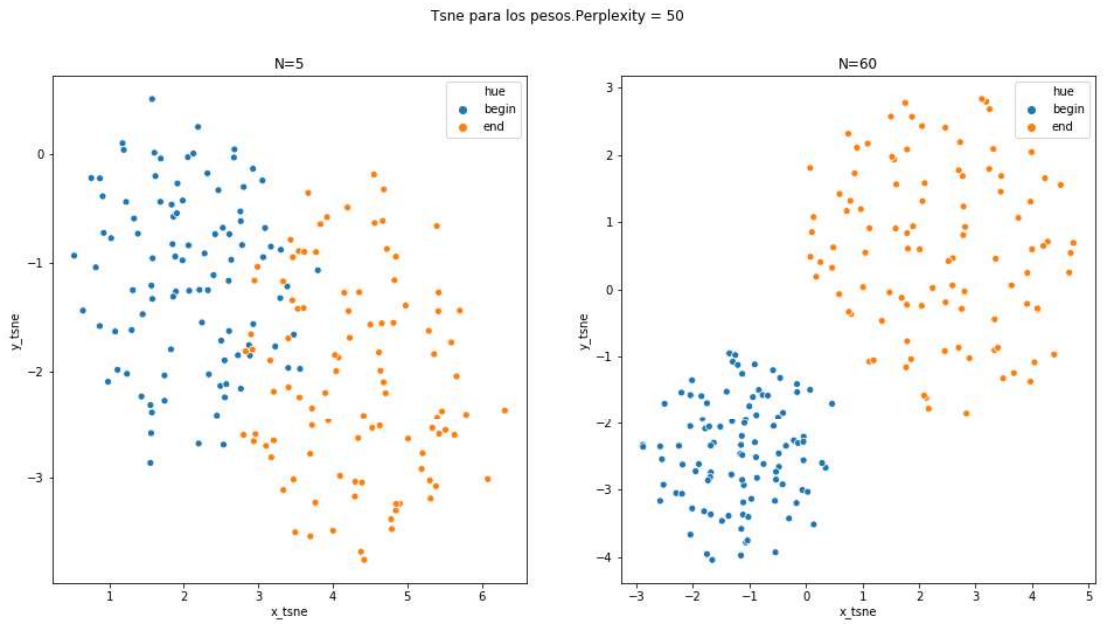

FIGURA 4.8: Resultados de t-SNE con Perplexity=50

Como podemos ver en los gráficos y tal dice la literatura, los resultados cambian considerablemente al modificarse el parámetro de *Perplexity*. A medida que lo vamos aumentando, el algoritmo de t-SNE diferencia mejor los pesos iniciales y finales, agrupándolos en diferentes regiones del espacio.

Utilizando esta herramienta, también pudimos graficar la trayectoria recorrida por el vector de pesos para cada Red desde la entrada hasta el entrenamiento como se puede ver en la figura 4.9. Para esto nos quedamos con el *Perplexity = 50* ya que este parece ser el parámetro que mejores resultados obtiene.

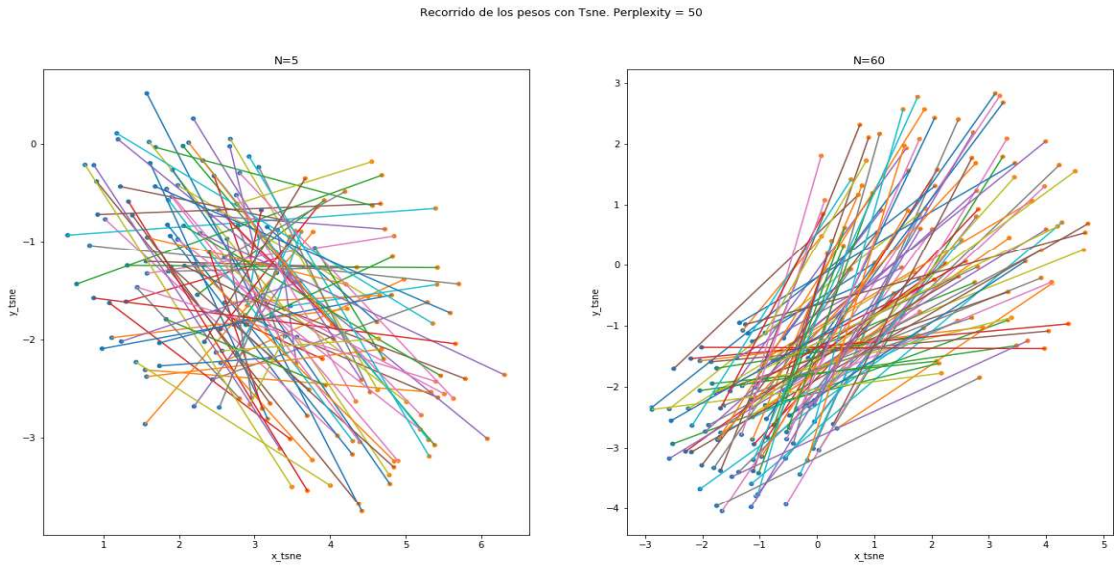

FIGURA 4.9: Trayectorias recorridas por los pesos para cada entrenamiento

Tanto para N=5 como para N=60 se pueden ver ciertos paralelismos en las trayectorias para puntos que están cerca en el espacio de salida. Esto podría indicar que si los pesos de inicio están en una misma región del espacio, luego de ser entrenados, van a otra misma región. Para N=5 se eligieron regiones del espacio de salida para graficar mejor estas direcciones paralelas como se ve en 4.10.

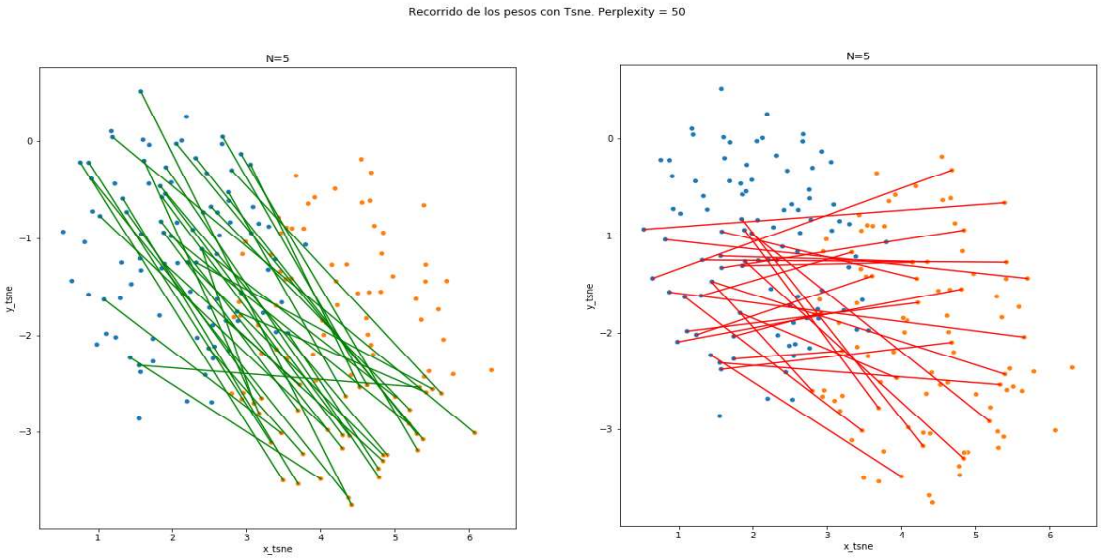

FIGURA 4.10: Trayectorias para sub-regiones del espacio

Los dos análisis que hicimos con los pesos de estas Redes, muestran que en ambos casos, tanto para la muestra N=5 como para N=60, la solución final se halla en lugares muy alejados del espacio de pesos. Es decir, tanto en un caso de interpolación (sobre-ajuste, pérdida nula) como en el caso de un resultado parsimonioso, los valores de los parámetros que codifican esas soluciones se hallan en lugares muy distintos del espacio. Por distintos entendemos que sus posiciones difieren entre sí más aún de lo que difieren sus posiciones iniciales, generadas al azar. Este resultado es compatible con el hecho que en una arquitectura sobre-parametrizda, es de esperar que existan infinitas combinaciones de pesos (ver 5) que expresan funciones similares.

Sin embargo, la pregunta que sigue abierta es: ¿Por qué con N=60 datos nunca se llega a la interpolación?.

Capítulo 5

Redes Random

La literatura, como hemos visto en el marco teórico de esta tesis, indica que existe una arquitectura que interpola los datos para todo N y algunos trabajos dan ciertas cotas para armar esa arquitectura según cantidad de neuronas y cantidad de capas pero ningún trabajo todavía da la arquitectura exacta. Esto nos hace pensar que existe la posibilidad de que la razón por la cual la Red no alcanza a interpolar los datos para N=60 es porque la arquitectura predefinida de forma aleatoria por nosotros no es capaz de codificar una función como la dada, luego para analizar esta hipótesis proseguimos con un análisis diferente.

En este caso, se definieron pesos uniformes (0,1) para asignarle a la red, y sin ningún tipo de entrenamiento, se calculó una función resultante con estos pesos aleatorios, evaluando la Red en una secuencia de longitud 1000 entre -2 y 2. Vamos a definir como $\bar{W}_a$ al vector de pesos aleatorios seleccionado. La función resultante se puede ver en la figura 5.1. De esta manera podemos asegurarnos que la función con la cual se van a generar los datos es una función codificable con una Red de esta arquitectura. Luego se seleccionaron de esta secuencia de forma aleatoria 60 datos para obtener nuestra muestra.

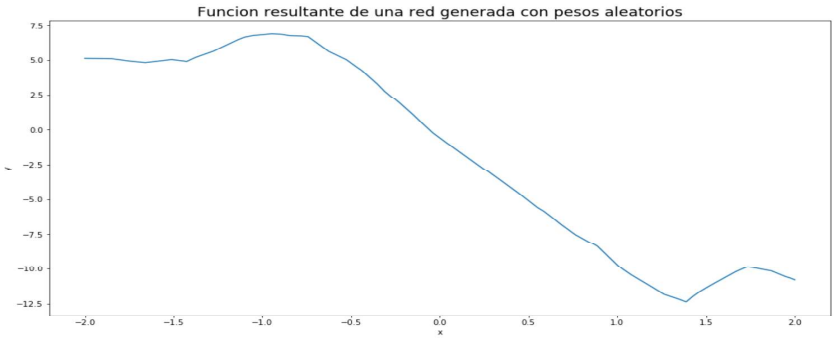

FIGURA 5.1: Función resultante de una Red con pesos random

Se corrieron 100 redes con pesos iniciales uniformes (0,1) (la misma distribución con la que se había generado $\bar{W}_a$), las predicciones obtenidas se pueden ver en la figura 5.1. Para estos entrenamientos se utilizo *learning rate* 0.001 y *epochs* 2000.

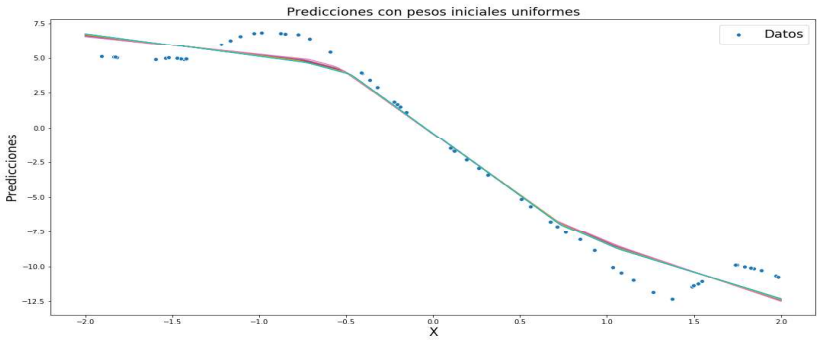

FIGURA 5.2: Predicciones obtenidas con pesos iniciales uniformes

A la hora de evaluar las pérdidas obtuvimos:

- Promedio de las pérdidas iniciales de la Red : 52.592
- Promedio de las pérdidas luego del entrenamiento de la Red : 1.401

Para los pesos, calculamos las métricas B, E y T 4.1, 4.2, 4.3, tomando S= 100:

- B= 2.37
- E= 5.20
- T= 3.53

Si recordamos la tabla 4.3 los valores obtenidos no difieren significativamente a los de la columna de N=60, eso quiere decir que el hecho de que la Red no llegue a interpolar los 60 datos no tiene nada que ver con que la función generadora de la muestra no sea un función codificable por este tipo de arquitectura.

En pos de lograr la interpolación de la muestra procedemos con el siguiente análisis. Ahora, en cada entrenamiento, se definieron pesos iniciales como $\bar{W}_a$ más un ruido uniforme en el intervalo $[-R, R]$ para ciertos R. Es decir, el objetivo de esto es analizar el comportamiento de la red, cuando el valor de entrada está 'cerca' del óptimo global en ese espacio de $\mathbb{R}^{1221}$. Así se busca observar como se mueve el Gradiente Descendente sobre la superficie de pérdida y dónde converge. Para cada valor de R se corrieron 10 entrenamientos.

Se utilizaron los siguientes diámetros R : 0.9, 0.7, 0.5, 0.2, 0.1, 0.05, 0.01. Para estos entrenamientos se utilizo *learning rate* 0.0001 y *epochs* 1000.

5.0.1. Predicciones resultantes y pérdida

En los siguientes gráficos se puede ver para cada R, la función codificada por los pesos antes y luego de ser entrenados.

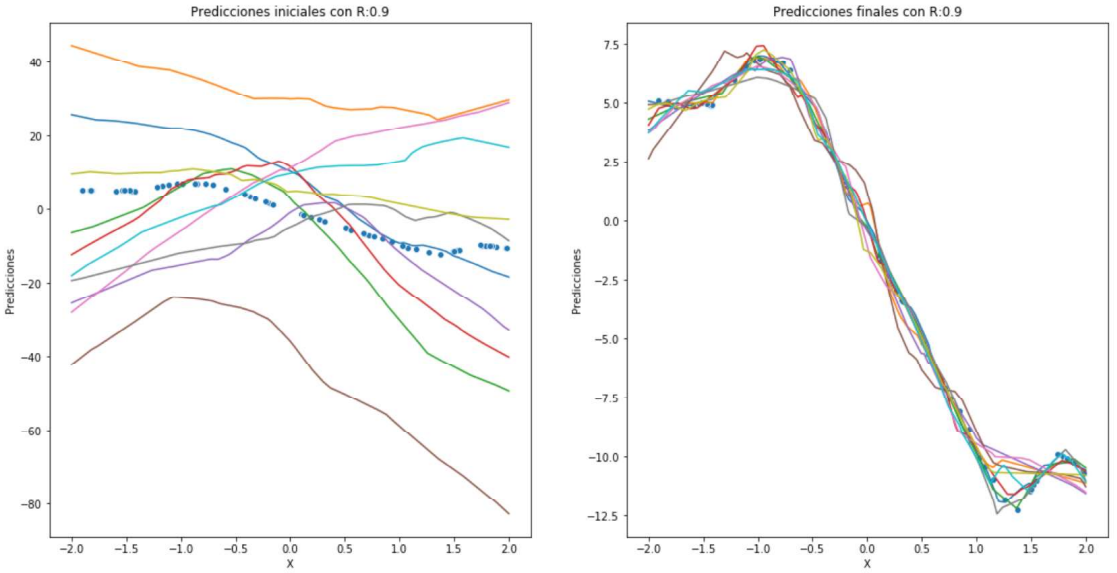

FIGURA 5.3: Predicciones obtenidos un R = 0.9

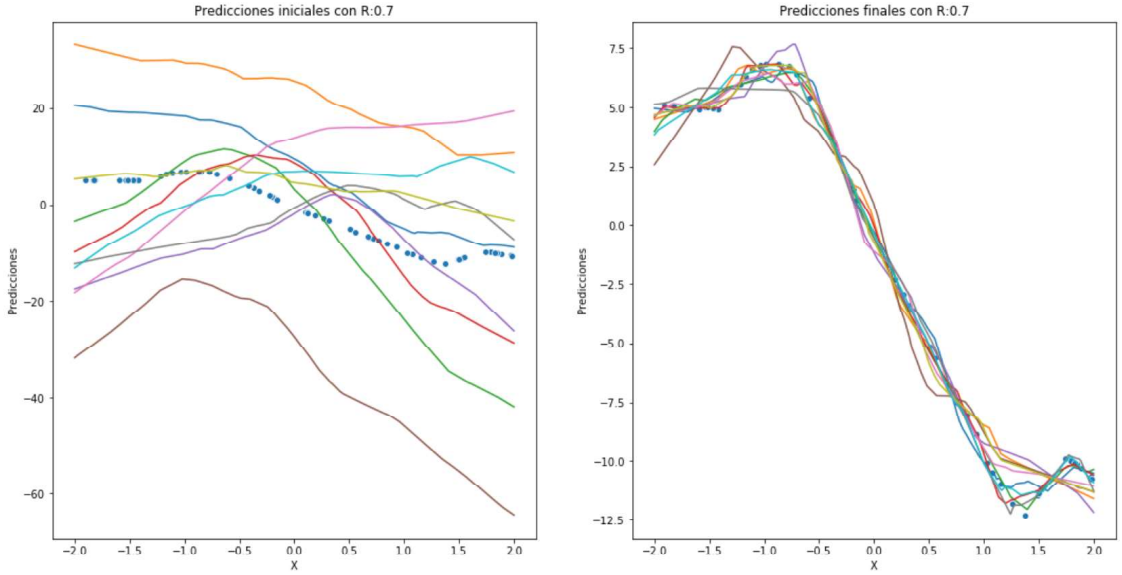

FIGURA 5.4: Predicciones obtenidos un R = 0.7

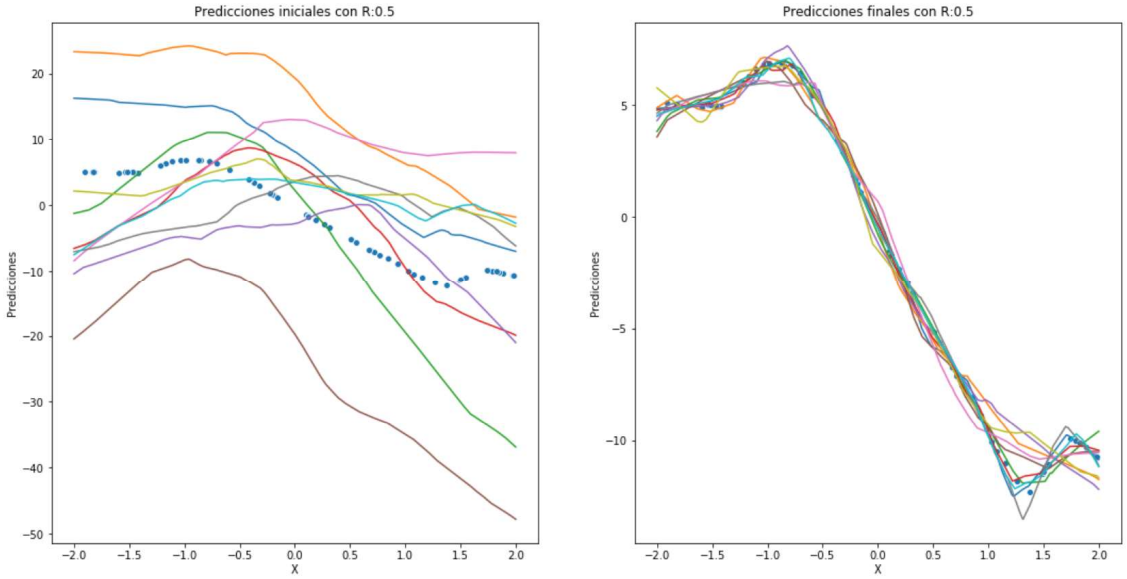

FIGURA 5.5: Predicciones obtenidos un R = 0.5

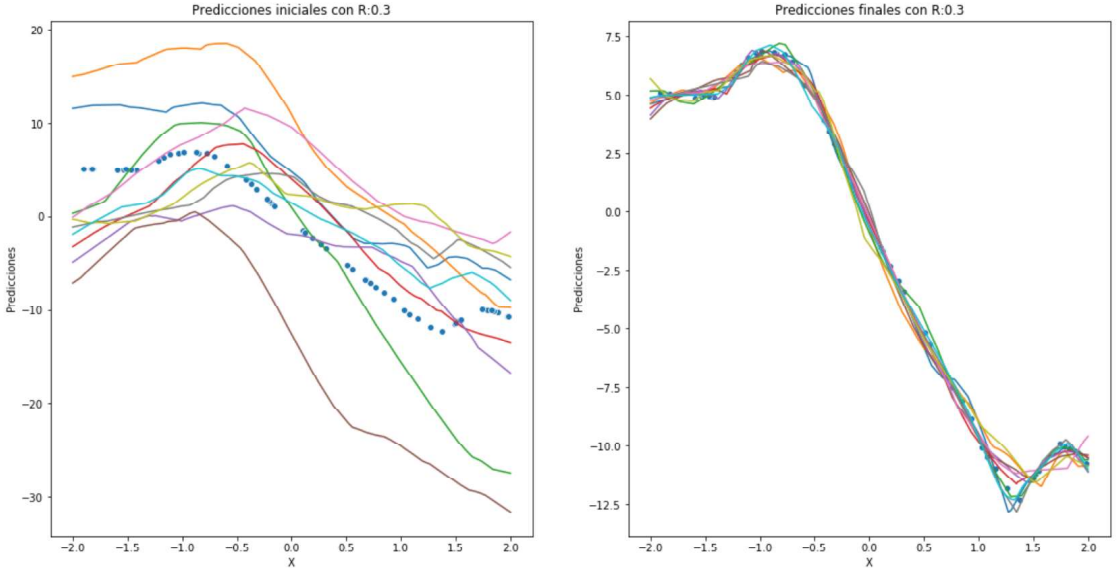

FIGURA 5.6: Predicciones obtenidos un R = 0.3

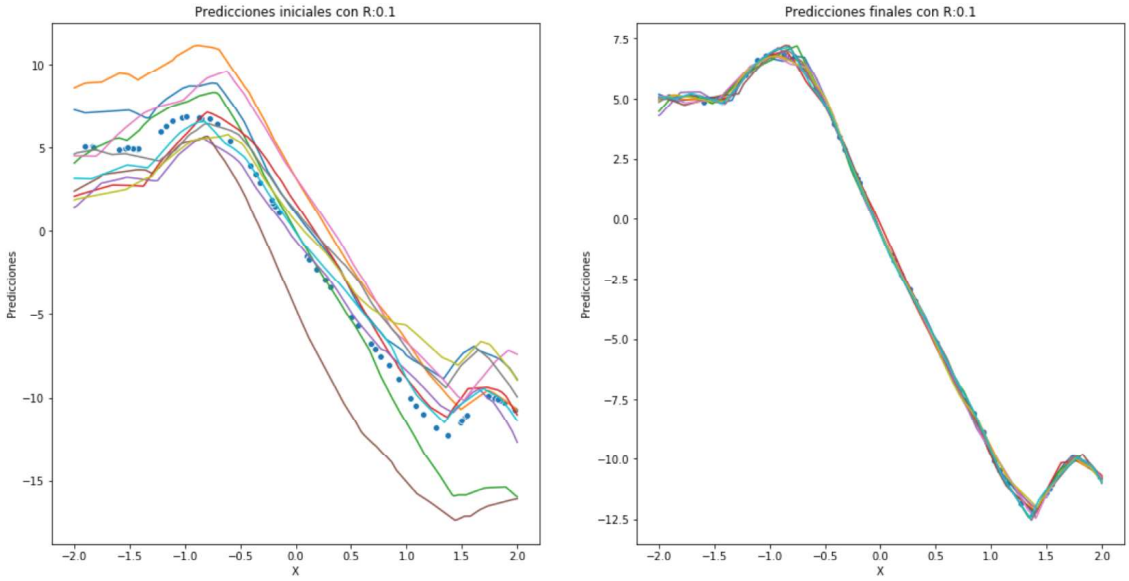

FIGURA 5.7: Predicciones obtenidos un R = 0.1

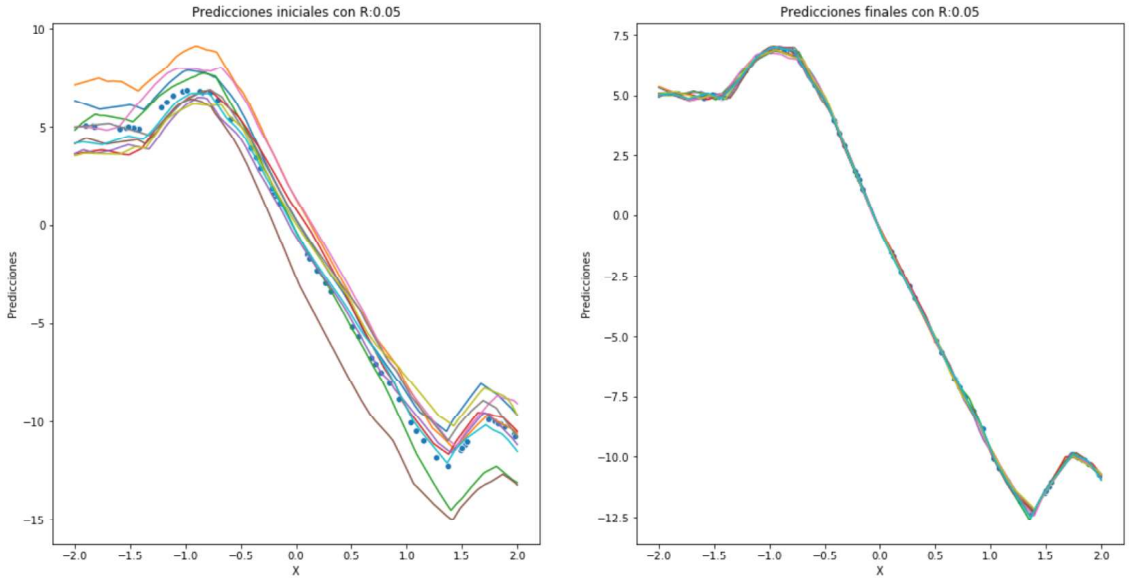

FIGURA 5.8: Predicciones obtenidos un R = 0.05

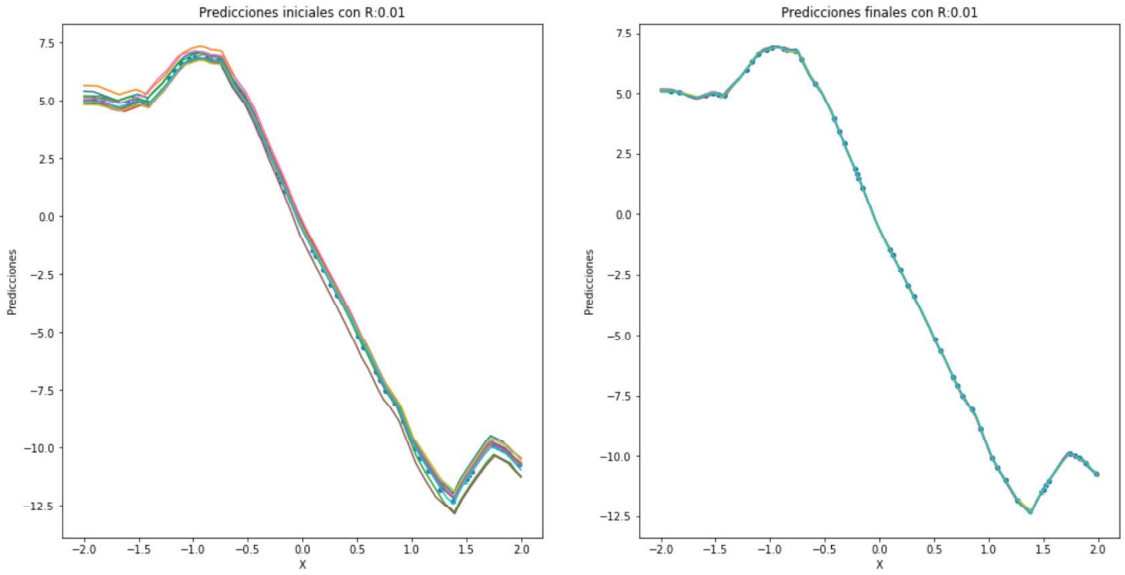

FIGURA 5.9: Predicciones obtenidos un R = 0.01

Para poder analizar las pérdidas, se calculó el ECM de cada entrenamiento, para la función final y para la función codificada inicialmente por los pesos seleccionados de forma uniforme, y luego se calculó un promedio entre los 10 entrenamientos de estos valores. Los resultados se pueden ver en la tabla 4.2. Agregamos los valores obtenidos previamente a la tabla también para poder comparar con las redes con pesos iniciales uniformes.

Radio de Bola (R)	Pérdida inicial	Pérdida final
Pesos uniformes	52.592	1.401
0.90	546.285	0.277
0.70	307.861	0.349
0.50	151.520	0.305
0.30	55.000	0.13
0.10	6.341	0.019
0.05	1.590	0.006
0.01	0.065	0.000

TABLA 5.1: Tabla de Pérdidas para cada diámetro R

Como podemos ver en la tabla y en los gráficos, a partir de R = 0.10, la pérdida inicial es menor a la pérdida obtenida con los pesos iniciales uniformes, pero en el caso de la pérdida resultante de los entrenamientos, para todo valor de R obtenemos mejores predicciones. O sea que finalmente nos vamos acercando a la interpolación de los datos. Además a medida que el R disminuye la pérdida final va disminuyendo, hasta que en el último caso R = 0.01, la Red entrenada interpola los datos dejando una pérdida final de 0. Ahora veamos si esta interpolación sucede debido a que el vector de pesos resultante del entrenamiento es el mismo $\bar{W}_a$. Es decir,

si la pérdida 0 se alcanza porque el algoritmo de optimización se movió en el espacio de 1221 alcanzando a $\bar{W}_a$ que es dónde la superficie tiene un mínimo global 0.

5.0.2. Métricas para los pesos

A las métricas antes definidas de B, E y T se agregaron dos métricas para este análisis:

$$W_b = \frac{1}{S}\sum_{i=1}^{S}\left\|\bar{W}_i^b - \bar{W}_a\right\|_2 \tag{5.1}$$

$$W_e = \frac{1}{S}\sum_{i=1}^{S}\left\|\bar{W}_i^e - \bar{W}_a\right\|_2 \tag{5.2}$$

En la tabla 5.2 podemos ver los resultados obtenidos, nuevamente agregamos el caso de las redes con pesos iniciales uniformes para poder comparar.

Radio de la Bola (R)	B	E	T	W_e	W_b
Pesos uniformes	2.371	5.203	3.537	36.123	36.324
0.90	25.736	25.737	0.832	18.219	18.212
0.70	20.017	20.011	0.715	14.170	14.166
0.50	14.298	14.282	0.574	10.122	10.112
0.30	8.579	8.564	0.398	6.073	6.063
0.10	2.860	2.849	0.151	2.024	2.017
0.05	1.430	1.424	0.077	1.012	1.008
0.01	0.286	0.284	0.016	0.202	0.201

Tabla 5.2: Tabla de Métricas para los pesos por cada diámetro R

La trayectoria recorrida por los pesos uniformes hasta converger es mayor a la recorrida para todo valor de R, y las distancias al vector inicial también lo cual tiene sentido ya que arrancamos de manera random. Además, W_e y W_b son similares para cada valor de R, esto quiere decir que la distancia que hay entre $\bar{W}_a$ y los pesos iniciales es comparable con la que hay con los pesos finales, o sea que podemos concluir que el gradiente descendente al entrenar, no alcanza $\bar{W}_a$ sino que se estanca en un vector que esta muy cerca donde evidentemente la superficie de pérdida tiene un mínimo local. Esto nos hace creer que la superficie de pérdida esta repleta de mínimos locales que inclusive sin importar que tan cerca estés del mínimo global, el algoritmo de optimización se va a terminar estancando en alguno de estos.

Capítulo 6

Redes interpoladoras

6.1. Función interpoladora

En esta sección vamos a cambiar el enfoque del análisis construyendo una Red interpoladora de los datos a partir de N datos (x, y).

Teorema 1. Sean (x_i, y_i) con $i = 1, ..., N$, N puntos de $\mathbb{R}^2$ tales que $x_1 < x_2 < ... < x_N$. Luego existe una función $F_N : \mathbb{R} \to \mathbb{R}$ que interpola los N datos, dada por:

$$F_N(x) = \alpha + P_1\ \sigma(x - x_1) + \sum_{i=2}^{N-1} (P_i - P_{i-1})\ \sigma(x - x_i) \tag{6.1}$$

donde σ refiere a la función de activación ReLU, $\alpha = y_1$ y P_i es la pendiente de la recta formada entre (x_i, y_i) y (x_{i+1}, y_{i+1}), es decir, $P_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}$.

Demostración. Recordemos que $\sigma(x) = max(0, x)$. Basta ver que $F_N(x_i) = y_i\ \forall\ i = 1, ..., N$. Vamos a verlo en el caso de x_1 y x_2 y luego para x_j con $j = 3, ..., N$.

- x_1: Reemplazando en 6.1,

$$F_N(x_1) = \alpha + P_1\ \sigma(x_1 - x_1) + \sum_{i=2}^{N-1} (P_i - P_{i-1})\ \sigma(x_1 - x_i)$$

  Como $x_1 \leq x_i\ \ \forall\ i = 1, ..., N \quad \Rightarrow \quad \sigma(x_1 - x_i) = 0\ \ \forall\ i = 1, ..., N$. Luego,

$$F_N(x_1) = y_1$$

- x_2:

$$F_N(x_2) = \alpha + P_1\ \sigma(x_2 - x_1) + \sum_{i=2}^{N-1} (P_i - P_{i-1})\ \sigma(x_2 - x_i)$$

  Análogamente, $x_2 \leq x_i\ \ \forall\ i = 2, ..., N \quad \Rightarrow \quad \sigma(x_2 - x_i) = 0\ \ \forall\ i = 2, ..., N$.

Además, $P_1 = \frac{y_2 - y_1}{x_2 - x_1}$. Luego,

$$F_N(x_2) = y_1 + \frac{y_2 - y_1}{x_2 - x_1} \ (x_2 - x_1) = y_2$$

- x_j con $j = 3, ..., N$:

$$F_N(x_j) = \alpha + P_1 \ \sigma(x_j - x_1) + \sum_{i=2}^{N-1} (P_i - P_{i-1}) \ \sigma(x_j - x_i)$$

Ahora dado j, como $x_j > x_i \quad , \quad \sigma(x_j - x_i) = \begin{cases} x_j - x_i & \forall \ i < j \\ 0 & \forall \ i \geq j \end{cases}$

Luego,

$$F_N(x_j) = y_1 + P_1 \ (x_j - x_1) + \sum_{i=2}^{j-1} (P_i - P_{i-1}) \ (x_j - x_i) \tag{6.2}$$

Centrémonos en el término de la sumatoria, $\sum_{i=2}^{j-1} (P_i - P_{i-1}) \ (x_j - x_i)$

$$\begin{aligned}
&= \sum_{i=2}^{j-1} P_i \ (x_j - x_i) - \sum_{i=2}^{j-1} P_{i-1} \ (x_j - x_i) \qquad \text{tomando } k = i - 1, \\
&= \sum_{i=2}^{j-1} P_i \ (x_j - x_i) - \sum_{k=1}^{j-2} P_k \ (x_j - x_{k+1}) \\
&= P_{j-1} \ (x_j - x_{j-1}) + \sum_{i=2}^{j-2} P_i \ (x_j - x_i) - \sum_{k=2}^{j-2} P_k \ (x_j - x_{k+1}) - P_1 \ (x_j - x_2) \\
&= -P_1(x_j - x_2) + \sum_{i=2}^{j-1} P_i(x_j - x_i - (x_j - x_{i+1})) + P_{j-1}(x_j - x_{j-1}) \\
&= -P_1(x_j - x_2) + \sum_{i=2}^{j-2} \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \ (x_{i+1} - x_i) + \frac{y_j - y_{j-1}}{x_j - x_{j-1}} \ (x_j - x_{j-1}) \\
&= -P_1(x_j - x_2) + \sum_{i=2}^{j-2} (y_{i+1} - y_i) + y_j - y_{j-1} \\
&= -P_1(x_j - x_2) + y_{j-1} - y_2 + y_j - y_{j-1} \\
&= -P_1(x_j - x_2) - y_2 + y_j
\end{aligned}$$

Volviendo a 6.2,

$$\begin{aligned}
F_N(x_j) &= y_1 + P_1 \ (x_j - x_1) - P_1(x_j - x_2) - y_2 + y_j \\
&= y_1 + \frac{y_2 - y_1}{x_2 - x_1}(x_2 - x_1) - y_2 + y_j \\
&= y_1 + y_2 - y_1 - y_2 + y_j \\
&= y_j
\end{aligned}$$

□

Teorema 2. Dados N puntos en $\mathbb{R}^2$, existe una Red Neuronal *fully connected* formada por una sola capa oculta de $N-1$ neuronas con activación ReLU que interpola los N datos.

Demostración. Dados N puntos, podemos calcular la función F_N definida por el teorema previo. Veamos como traducir los coeficientes de F_N a los parámetros de una Red con una capa y $N-1$ neuronas. Sean b_i, W_i^{IN} y W_i^{OUT} el bias y los pesos de la $i-esima$ neurona de entrada y de salida respectivamente, y por último B_O el bias de la capa de neuronas. Entonces, la función resultante de la Red con esta arquitectura y activación ReLU la podemos escribir de la siguiente manera:

$$G(x) = \sum_{i=1}^{N-1} W_i^{OUT} \sigma(W_i^{IN} x + b_i) + B_O$$

Si retomamos la expresión de F_N en 6.1, basta ver que $G \equiv F_N$ tomando:

- $B_0 = y_1$
- $b_i = -x_i \quad \forall\ i = 1, ..., N-1$
- $W_i^{IN} = 1 \quad \forall\ i = 1, ..., N-1$
- $W_1^{OUT} = P_1$
- $W_i^{OUT} = P_i - P_{i-1} \quad \forall\ i = 2, ..., N-1$

□

6.1.1. Observaciones de la función interpoladora

Como primera observación, debemos mencionar que mientras que la función de activación sea lineal, la función interpoladora trivialmente no es única ya que al parámetro W_i^{OUT} podemos multiplicarlo por una constante k_i y dividir por la misma a los parámetros W_i^{IN} y b_i y así obtener la misma función resultante. Es decir que si bien en el teorema hallamos una combinación de parámetros que genera una función interpoladora de los datos, esta combinación no es única y existen infinitas funciones interpoladoras.

Tal vimos en el *Teorema 1*, dados N datos existe una Red formada por una capa oculta de N-1 neuronas con activación ReLU que logra interpolar los datos. Ahora como segunda observación vamos a analizar qué pasaría si le agregáramos una neurona a esa capa, ¿existe una Red con N neuronas que interpole N datos?

Retomando el ejemplo anterior, sean (x_i, y_i) con $i = 1, ..., N$ tales que $x_1 < x_2 < ... < x_N$, definimos una Red formada por N neuronas con activación ReLU. La función resultante de la Red es:

$$G(x) = \sum_{i=1}^{N} W_i^{OUT} \sigma(W_i^{IN} x + b_i) + B_O$$

Vamos a escribirla como:

$$G(x) = F(x) + W_N^{OUT} \sigma(W_N^{IN} x + b_N)$$

donde F(x) es la función formada por N-1 neuronas que interpola los N datos (6.1). Luego, al agregar una neurona tenemos 3 parámetros nuevos de la red, por simplicidad los vamos a llamar (W_{in}, b, W_{out}). Veamos que valores pueden tener estos para que la Red resultante siga interpolando los N datos.

Como F(x) está definida de forma tal que $F(x_i) = y_i \quad \forall i = 1, ..., N$, debemos hallar ($W_{in}$, b, W_{out}) tal que:

$$W_{out} \quad \sigma(W_{in} x + b) = 0 \quad \forall i \quad = 1, ..., N$$

Se puede ver, en principio, que no existe una solución única para esto. Veamos como calcular todas las soluciones:

- $W_{out} = 0$ para cualquier valor de W_{in} y b satisface lo pedido
- $\sigma(W_{in}\text{x} + \text{b}) = 0 \Leftrightarrow W_{in} x_i + b \leq 0 \quad \forall \quad i = 1, ..., N$

  Para despejar esto separamos en dos casos:

  1. $W_{in} \geq 0 \rightarrow x_i \leq \frac{-b}{W_{in}} \quad \forall \quad i = 1, ..., N$
     como $x_1 < x_2 < ... < x_N$, entonces basta con que $x_N \leq \frac{-b}{W_{in}}$
  2. $W_{in} \leq 0 \rightarrow x_i \geq \frac{-b}{W_{in}} \quad \forall \quad i = 1, ..., N$
     como $x_1 < x_2 < ... < x_N$, entonces basta con que $x_1 \geq \frac{-b}{W_{in}}$

Si consideramos a (W_{in}, b, W_{out}) como un espacio de 3 dimensiones, entonces la región del espacio donde se encuentran las soluciones de este problema lo podemos escribir de la siguiente forma:

$$R = R_1 \cup R_2 \cup R_3$$

donde,

$R_1 = \{W_{out} = 0 : W_{in}, b \in \mathbb{R}\}$

$R_2 = \{W_{in} x_N + b \leq 0 : W_{in} \geq 0, b, W_{out} \in \mathbb{R}\}$

$R_3 = \{W_{in} x_N + b \geq 0 : W_{in} \leq 0, b, W_{out} \in \mathbb{R}\}$

Como en este caso el espacio es de dimensión 3 podemos graficar estos espacios. En el caso de R_1 graficamente obtenemos un hiperplano que pasa por el (0,0,0) y tiene normal paralela al eje de W_{out}. En la figura 6.1 podemos ver como queda el gráfico de R_2 (el caso de R_3 es análogo a este) para el ejemplo que teníamos donde $x \in [-2, 2]$, para simplificar el gráfico tomamos x_1 = -2 y $x_N = 2$. En el gráfico de la proyección sobre $\{W_{out} = k\}$ definimos $r1 : W_{in} = 0$, $r2 : 2.W_{in} + b = 0$.

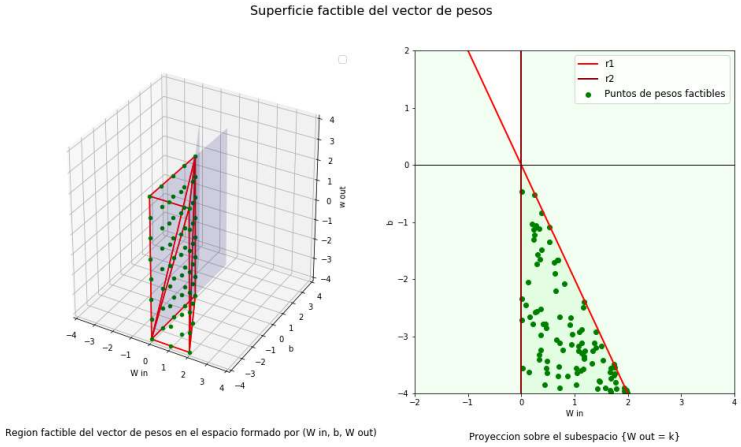

FIGURA 6.1: Gráfico de R_2

Tanto en los cálculos como en los gráficos podemos ver entonces que con tan solo agregar 1 neurona, estos nuevos 3 parámetros de la Red pueden tomar infinitos valores de forma tal que la Red resultante interpole los N datos.

Luego, existen infinitas redes formadas por una sola capa con N neuronas con activación ReLU que interpolar N datos, la función de pérdida entonces tiene infinitos mínimos globales. El mismo análisis se podría hacer agregando no una sino dos neuronas a la capa, el espacio va a ser de dimensión 6 y podemos inferir que la cantidad de mínimos globales va a ir aumentando, complejizando esto la superficie de pérdida y la optimización de la función de pérdida.

6.1.2. Ejemplo de Red interpoladora para 60 datos

La idea de esta sección es observar el nivel de efectividad que tiene el método de GD para una Red de una capa de neuronas cuando se inicializa el algoritmo en un entorno suficientemente cercano al óptimo global. Si generamos pesos iniciales en una bola de radio R centrada en el vector de pesos que interpola los puntos, la intuición indica que conforme achicamos el radio de dicha bola mayor será la probabilidad que el vector de pesos final se corresponda con una solución de pérdida 0. Veamos que es lo que ocurre mediante un ejemplo.

Sea ahora $y = f(x) + z/20$, y generamos N=60 realizaciones tomando $f(x) = 3 + \left(x - \frac{1}{2}\right)^2$, $x \sim U(0, 1)$ y $z \sim N(0, 1)$. De esta forma tenemos N=60 pares de valores (x, y) que procedemos a normalizarlos restándoles la media y dividiendo por el desvío para obtener nuestra muestra.

Definimos una Red *fully-connected* de una capa con 59 neuronas. Como podemos ver en la figura 6.2, la arquitectura resultante esta compuesta por 178 parámetros.

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 59)	118
dense_1 (Dense)	(None, 1)	60

Total params: 178
Trainable params: 178
Non-trainable params: 0

FIGURA 6.2: Summary del Modelo con 59 neuronas

Utilizando el *Teorema 2* calculamos los parámetros de la Red para interpolar nuestra muestra de 60 datos, llamemos a este vector $\bar{W}_I$, recordemos $\bar{W}_I \in \mathbb{R}^{178}$ y que no es el único vector en este espacio que interpola la muestra. Habiendo definido la Red, la evaluamos en una secuencia entre [-2,2] y obtuvimos el gráfico de la figura 6.3.

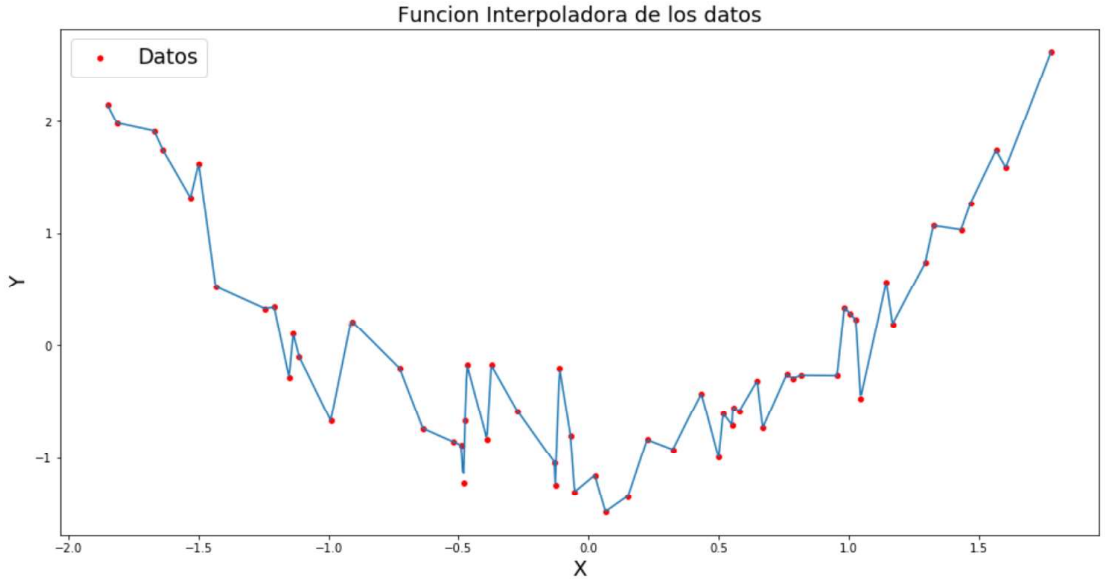

FIGURA 6.3: Red Interpoladora de 59 neuronas

Ahora vamos a tomar como pesos iniciales de la Red, vectores aleatorios generados en una bola de radio R centrada en el vector de pesos $\bar{W}_I$, y entrenamos 10 redes para cada caso. En este análisis, hay que tener en cuenta que los pesos iniciales tienen magnitudes más grandes que en los casos anteriores. Esto se debe principalmente a que los W_{OUT} están formados por las pendientes de las rectas que unen los puntos o la resta de estas pendientes, por ende dependiendo de la muestra que uno le de, estos pesos pueden ser tan grandes como uno quiera. Luego para proceder con los entrenamientos de las Redes de forma exitosa, se modificó el parámetro el *learning rate* 2,2, tomando $\gamma = 0{,}00003$ fijo en todas las iteraciones. Todos los entrenamientos además fueron realizados con 1000 *epochs*.

Tomamos R como 0.9, 0.5, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001, 0.00005, 0.00001 y en los siguientes gráficos se pueden ver los resultados obtenidos.

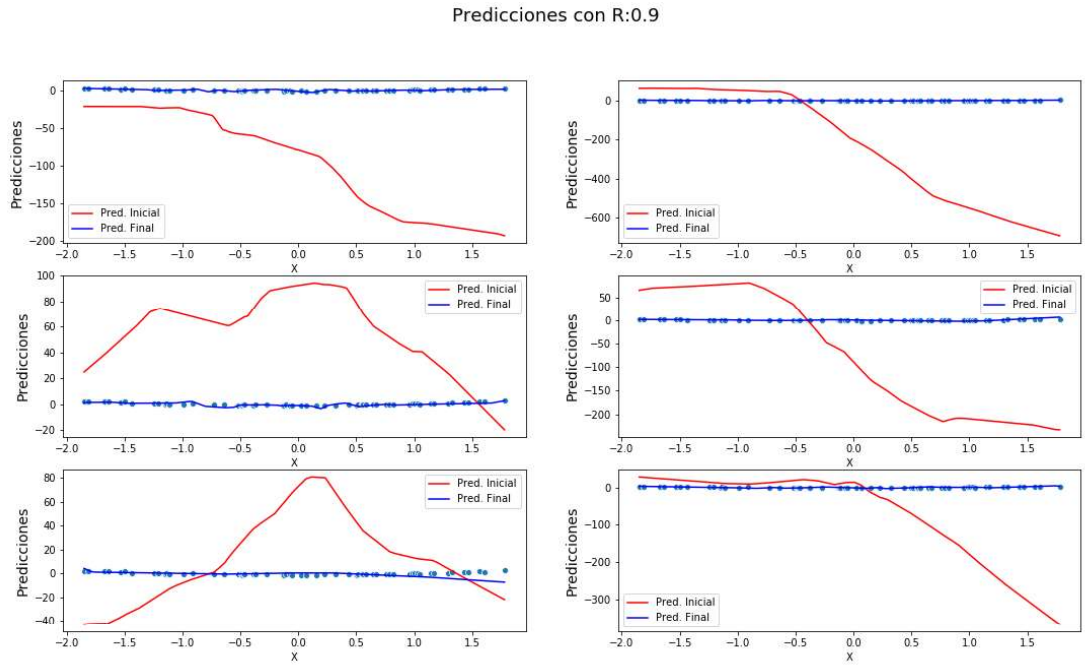

FIGURA 6.4: Predicciones obtenidos con R = 0.9

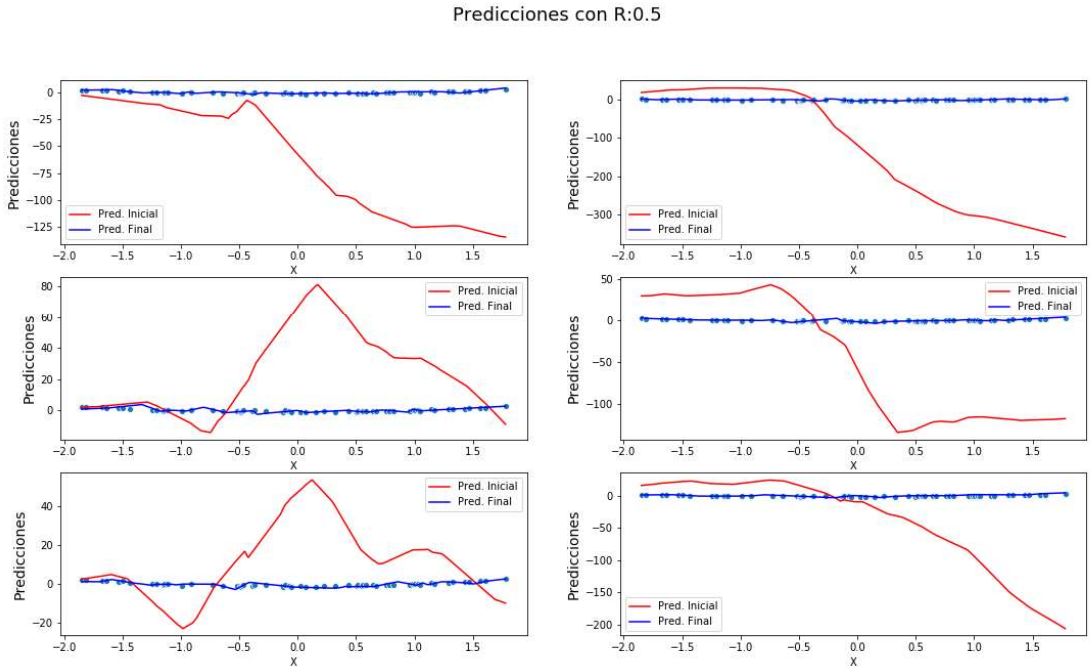

FIGURA 6.5: Predicciones obtenidos con R = 0.5

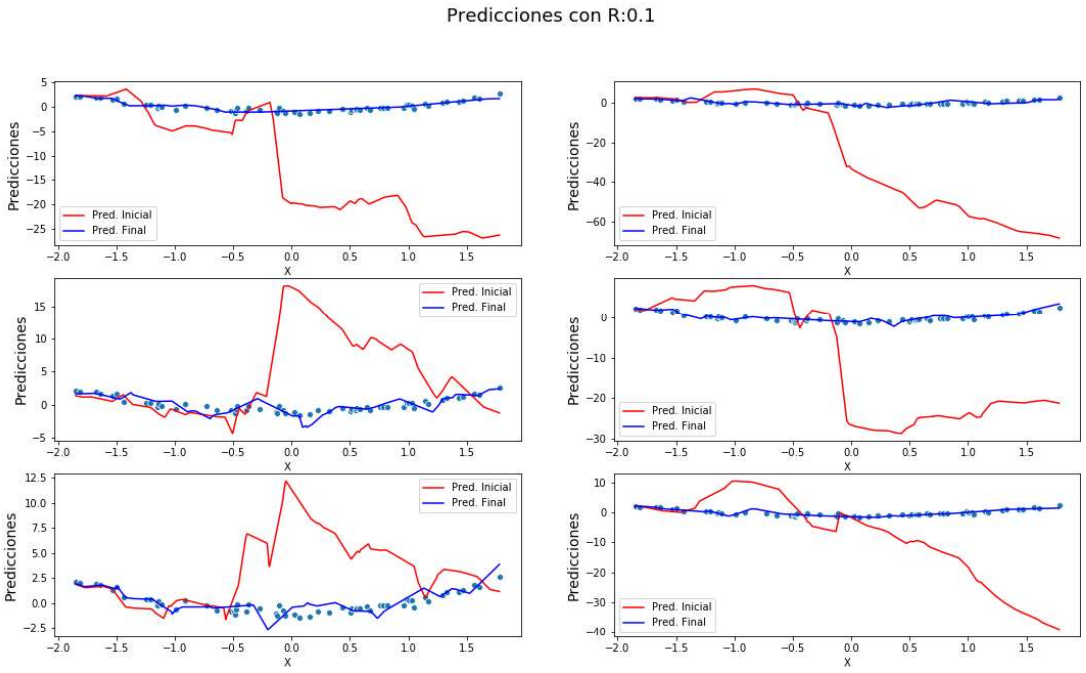

FIGURA 6.6: Predicciones obtenidos con R = 0.1

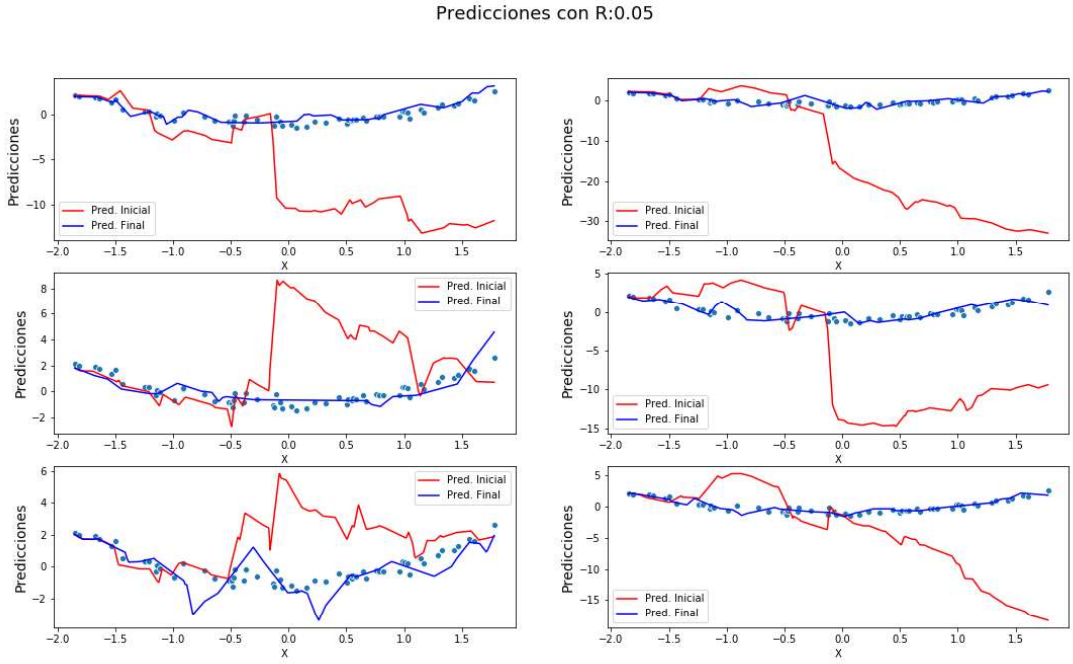

FIGURA 6.7: Predicciones obtenidos con R = 0.05

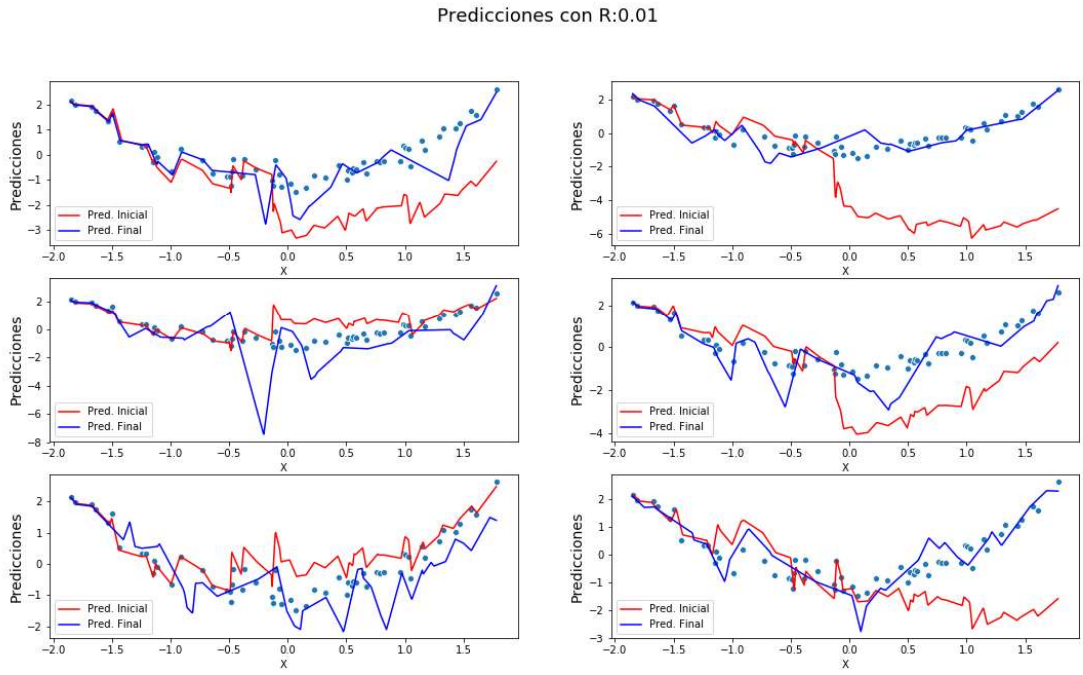

FIGURA 6.8: Predicciones obtenidos con R = 0.01

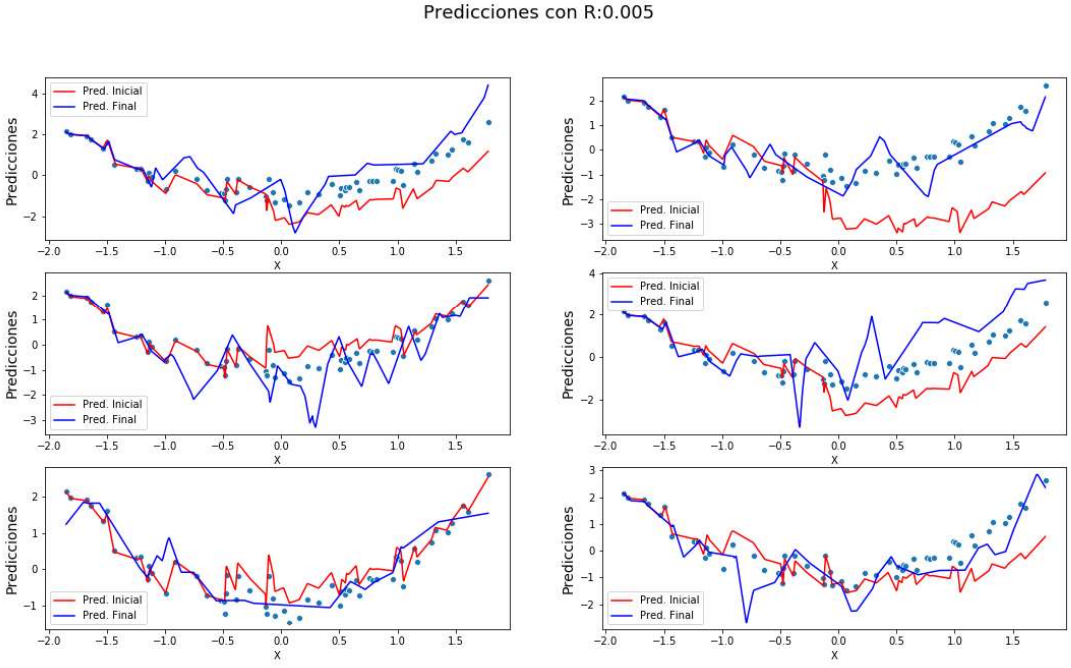

FIGURA 6.9: Predicciones obtenidos con R = 0.005

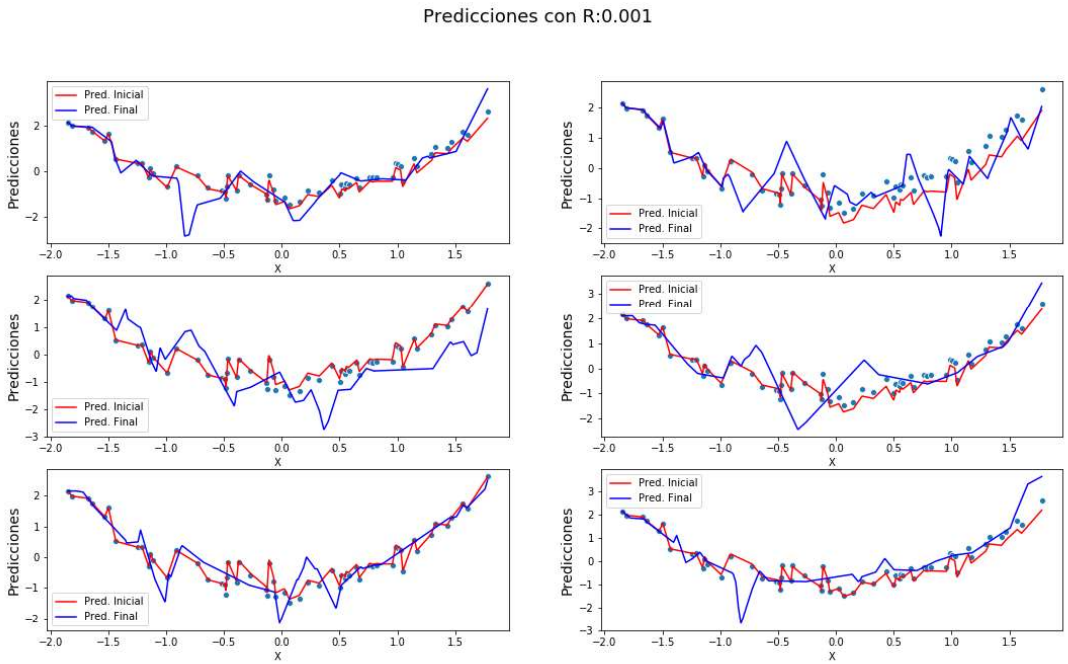

FIGURA 6.10: Predicciones obtenidos con R = 0.001

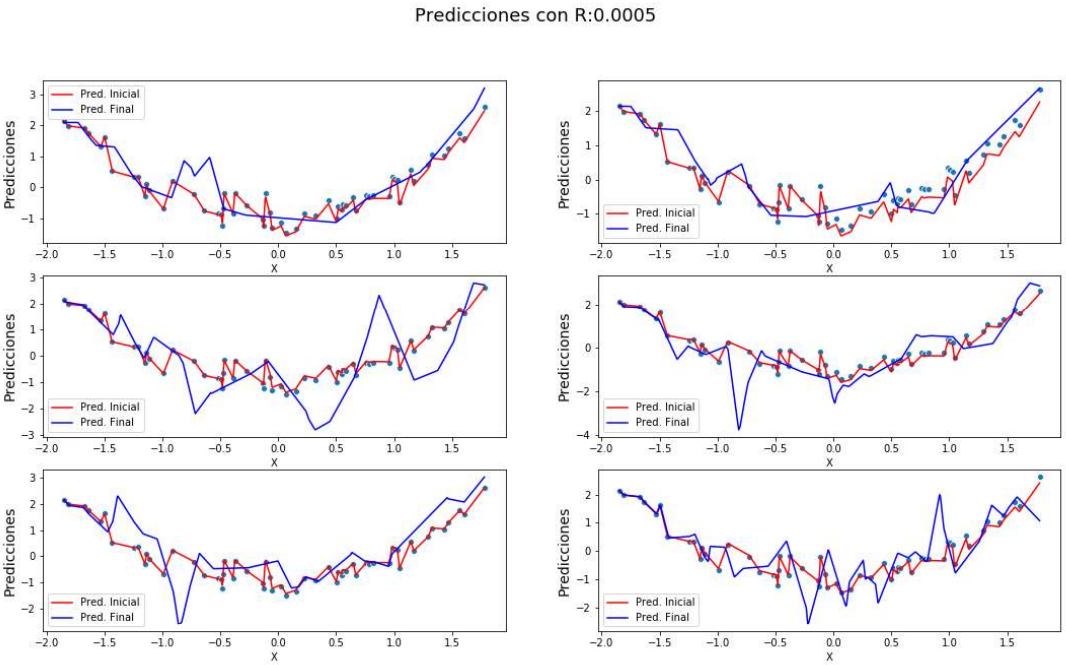

FIGURA 6.11: Predicciones obtenidos con R = 0.0005

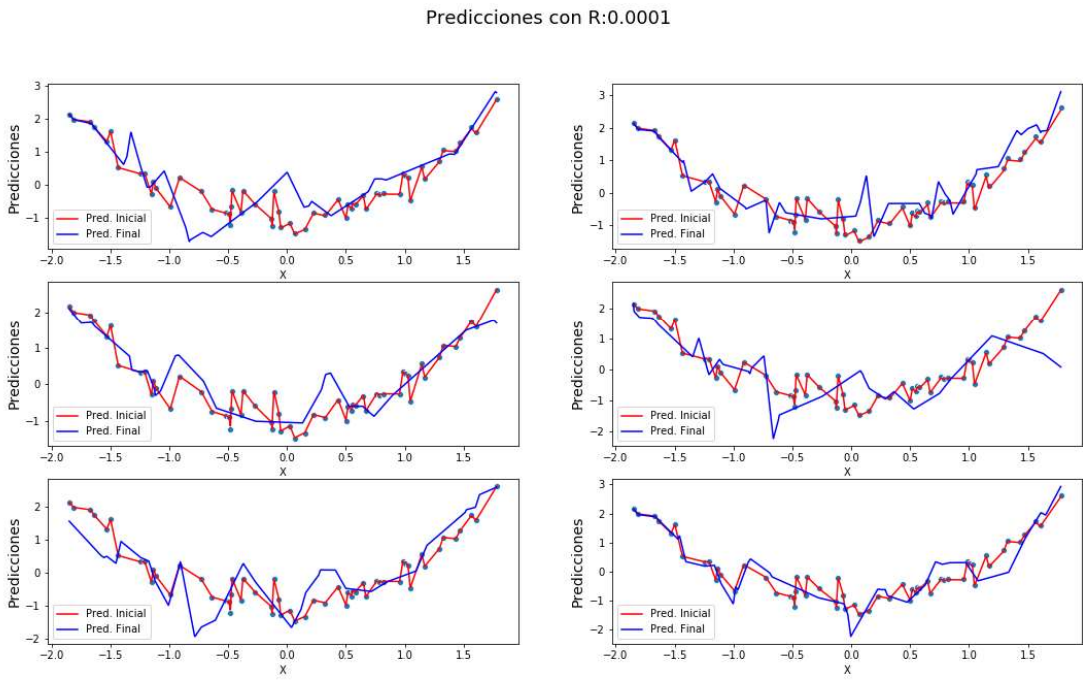

FIGURA 6.12: Predicciones obtenidos con R = 0.0001

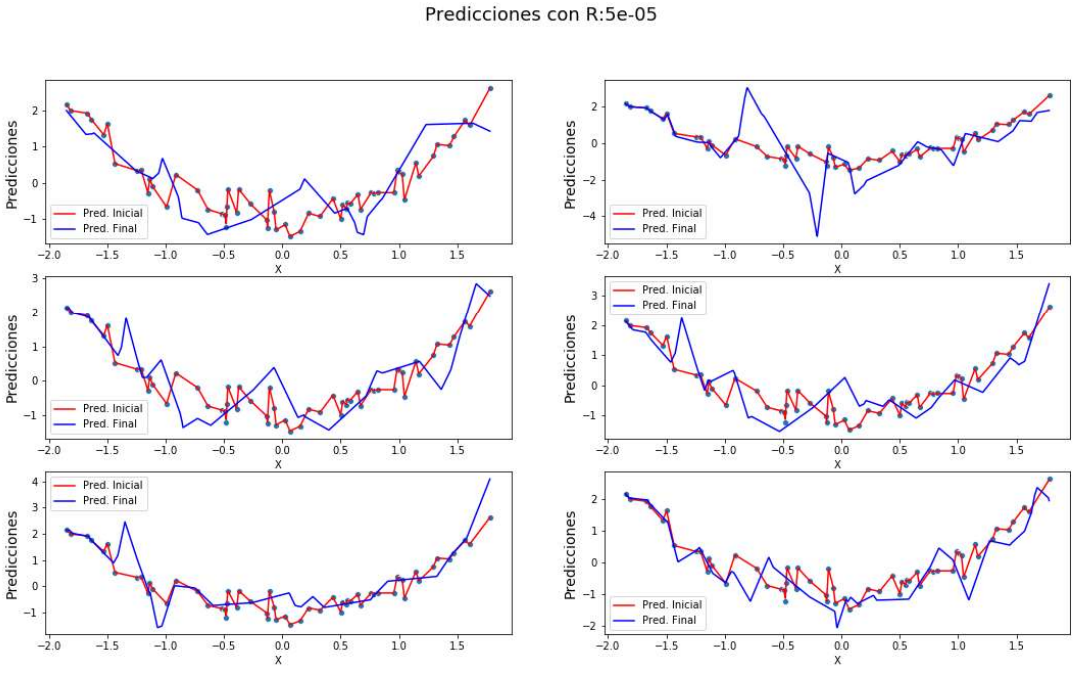

FIGURA 6.13: Predicciones obtenidos con R = 0.00005

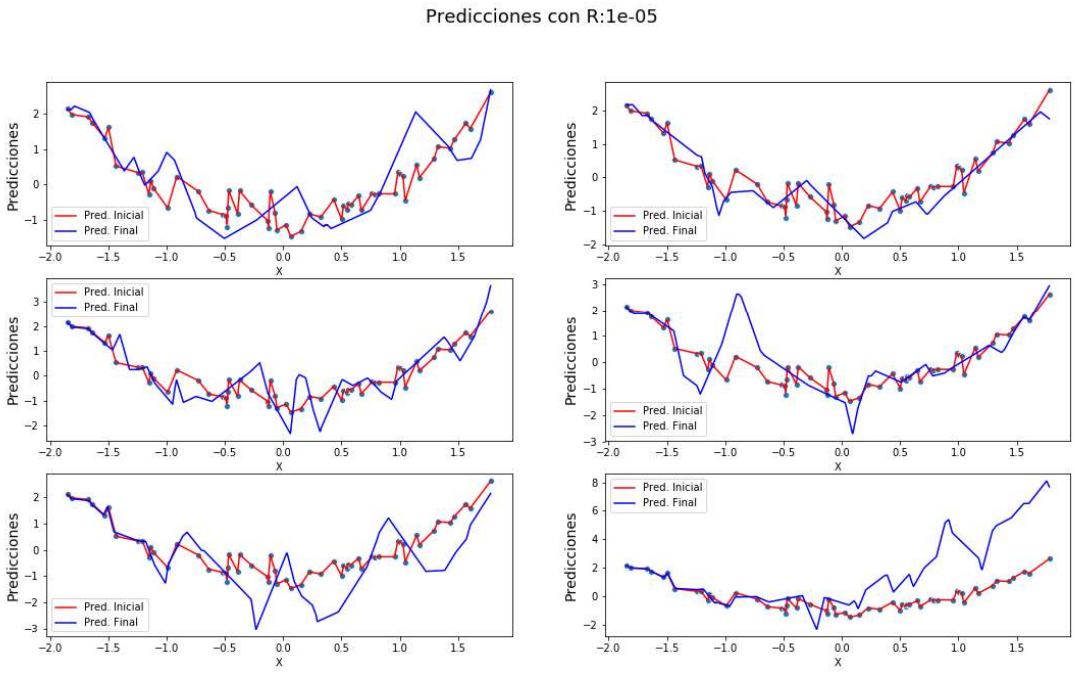

FIGURA 6.14: Predicciones obtenidos con R = 0.00001

Si bien para los valores más grandes de R parece que la Red resultante es más parsimoniosa que para los más chicos esto se debe solo a una cuestión de escala. En todos los casos de R, la función a la que se llega luego de entrenar la Red es a una función continua a tramos con muchos puntos de no derivabilidad, como podemos ver en la figura 6.15 que compara para algunos valores de R las redes finales. Ni siquiera considerando los R más chicos luego de entrenar llegamos a la interpolación, es decir, a la pérdida 0. Esto nos indica que sin importar que tan cerca estemos originalmente de los pesos interpoladores y de la pérdida 0, el gradiente descendente se mueve en el espacio de $\mathbb{R}^{178}$ y encuentra un mínimo local lo suficientemente bueno como para estancarse ahí. La superficie de pérdida evidentemente esta repleta de mínimos locales donde en cada entrenamiento el algoritmo de optimización se detiene.

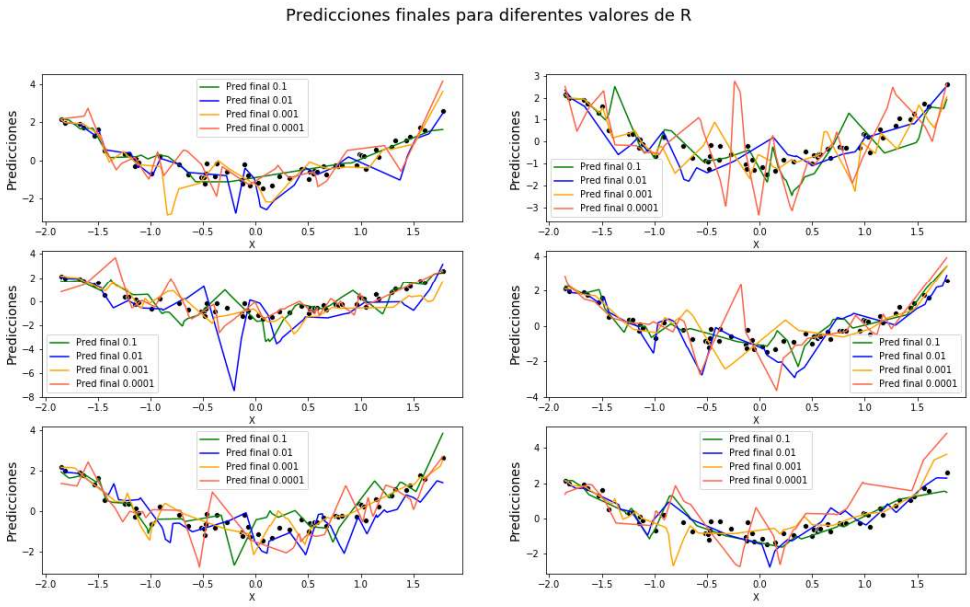

FIGURA 6.15: Predicciones finales con varios valores de R

En las figuras 6.16, 6.17 y 6.18 podemos ver como avanza la pérdida de entrenamiento de una Red para cada valor de R. En estos entrenamientos para los diferentes valores de R lo que podemos ver es que en los primeras iteraciones la pérdida explota y luego a medida que el algoritmo avanza va mejorando. Esto se lo adjudicamos al hecho de que al estar calculados exactamente los pesos interpoladores, un cambio en estos pesos, aunque sea del orden de 0.1, modifica todas las pendientes de las rectas generadas entre punto y punto y las predicciones por ende quedan muy dispares. Pero en todos los casos de R, destacamos que el algoritmo logra aproximarse a la función para obtener pérdidas finales similares y relativamente buenas, en comparación de las pérdidas iniciales, como podemos ver en la 6.1.

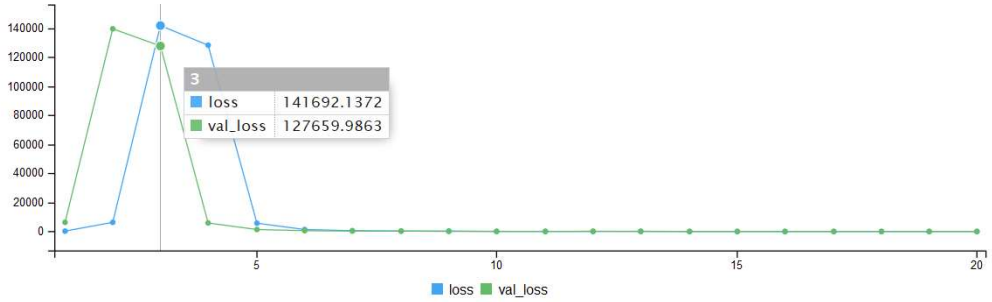

FIGURA 6.16: Training loss en un entrenamiento de R = 0.1

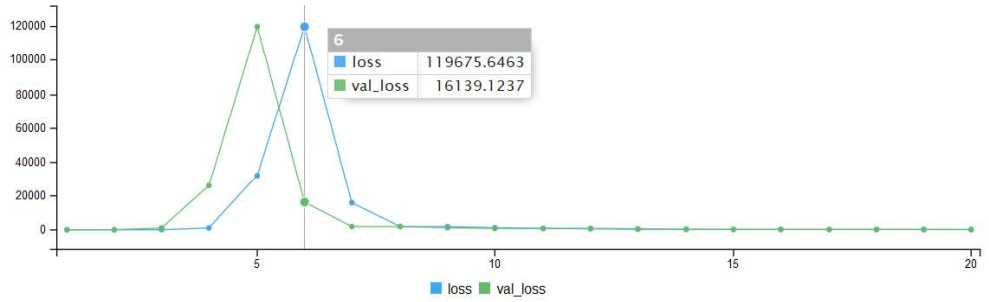

FIGURA 6.17: Training loss en un entrenamiento de R = 0.001

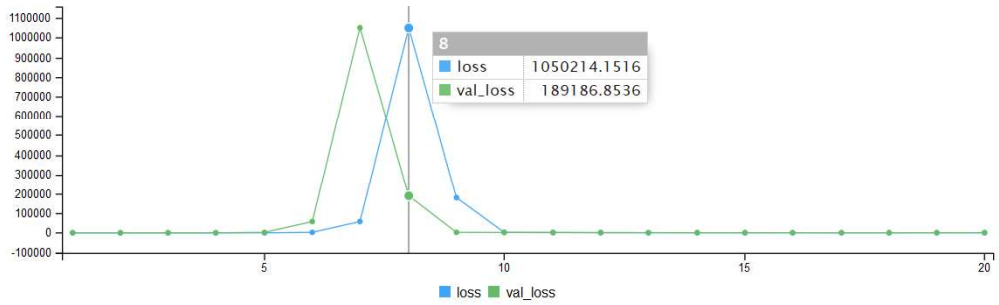

FIGURA 6.18: Training loss en un entrenamiento de R = 0.00001

En la tabla 6.1 podemos ver que la pérdida para R 0.9 es considerablemente mayor que en los otros casos, por lo tanto para valores menores de R podemos concluir que la predicción final es igual de 'buena', como también se ve en los gráficos.

Radio de Bola (R)	Pérdida inicial	Pérdida final
0.9	25076.39063	1.49849
0.5	8047.36923	0.44587
0.1	323.28150	0.41404
0.05	81.09643	0.38325
0.01	3.27273	0.55110
0.005	0.81876	0.41367
0.001	0.03275	0.34866
0.0005	0.00819	0.32650
0.0001	0.00033	0.87773
0.00005	0.00008	0.37981
0.00001	0.00000	0.81560

TABLA 6.1: Tabla de Pérdidas para cada diámetro R

Vamos a ver las métricas obtenidas por los pesos para ver como se movieron en este espacio de $\mathbb{R}^{178}$ en los entrenamientos. Análogamente al análisis anterior, tomamos:

$$W_b = \frac{1}{10}\sum_{i=1}^{10}\left\|\bar{W}_i^b - \bar{W}_I\right\|_2 \tag{6.3}$$

$$W_e = \frac{1}{10}\sum_{i=1}^{10}\left\|\bar{W}_i^e - \bar{W}_I\right\|_2 \tag{6.4}$$

R	B	E	T	W_b	W_e
0.9	8.687	81.480	50.462	6.918	52.213
0.5	4.826	13.044	12.288	3.843	13.046
0.1	0.965	22.325	22.557	0.768	22.588
0.05	0.482	13.142	15.310	0.384	15.318
0.01	0.096	15.444	17.587	0.076	17.588
0.005	0.048	8.289	10.385	0.038	10.385
0.001	0.009	10.585	13.002	0.007	13.003
0.0005	0.004	13.069	16.642	0.003	16.642
0.0001	0.001	25.479	22.140	0.0008	22.140
0.00005	0.0005	11.509	13.906	0.0004	13.906
0.00001	0.0001	12.370	12.675	0.0001	12.675

Tabla 6.2: Tabla de Métricas para los pesos por cada diámetro R

Para todos los valores de R las trayectorias recorridas parecen ser similares, lo mismo pasa como en los análisis anteriores entre la distancia que hay del $\bar{W}_I$ a los pesos iniciales como a los finales. En todos los casos, los pesos finales no son los interpoladores, lo que va de acuerdo a que el gradiente alcanza un mínimo local, lo suficientemente bueno para la pérdida y se detiene ahí sin alcanzar el mínimo global.

Ahora repetimos el análisis pero modificando los hiperparámetros del algoritmo de optimización de las redes. Definimos un *learning rate* más chico todavía, de 0.000005 y aumentamos las *epochs* a 2000, al modelo resultante lo vamos a llamar Modelo B. Vamos a comparar los valores obtenidos con los resultantes de los parámetros anteriores (*learning rate* 0.00003 y *epochs* 1000), al que llamaremos Modelo A.

R	Perdida inicial	Perdida final A	Perdida final B
0.9	25076.390	1.49849	3.33417
0.5	8047.36923	0.44587	2.78549
0.1	323.28150	0.41404	0.38819
0.05	81.09643	0.38325	0.10654
0.01	3.27273	0.55110	0.00561
0.005	0.81876	0.41367	0.00200
0.001	0.03275	0.34866	0.00004
0.0005	0.00819	0.32650	0.00003
0.0001	0.00033	0.87773	0
0.00005	0.00008	0.37981	0
0.00001	0	0.81560	0

Tabla 6.3: Tabla de Pérdidas por cada diámetro R

Como podemos ver en 6.3, para los valores de R mayores, el modelo B obtiene resultados peores, esto creemos se debe a que en esos valores de R, los hiperparámetros del Modelo A le dan mayor libertad para el algoritmo optimizador para moverse en la superficie de pérdida y alcanzar una mejor solución parsimoniosa (aunque lejos este de la óptima). En el caso del Modelo B, el paso es muy chico e inicialmente el vector inicial está demasiado lejos como para que pueda alcanzar al mínimo global sin importar de las iteraciones que se le de. Esta comparación cambia al analizar R más chicos, a partir del R 0.1, el Modelo B le gana a en todos los valores de R al Modelo A. Podemos inferir que estando lo suficientemente cerca, si al algoritmo se le dan las iteraciones que necesita con un paso lo suficientemente (infinitamente) chico puede acercarse al mínimo global.

R	B	E Mod A	E Mod B	T Mod A	T Mod B
0.9	8.6878	81.4808	8.6125	50.4628	1.0335
0.5	4.8266	13.0442	4.7907	12.2882	0.7258
0.1	0.9653	22.3257	0.9351	22.5579	0.1785
0.05	0.4827	13.1426	0.4606	15.3103	0.0957
0.01	0.0965	15.4443	0.0902	17.5879	0.0214
0.005	0.0483	8.2892	0.0448	10.3856	0.0110
0.001	0.0097	10.5859	0.0089	13.0029	0.0023
0.0005	0.0048	13.0697	0.0045	16.6427	0.0011
0.0001	0.0010	25.4799	0.0009	22.1407	0.0002
0.00005	0.0005	11.5093	0.0005	13.9061	0.0001
0.00001	0.0001	12.3707	0.0001	12.6756	0.0000

Tabla 6.4: Tabla de Métricas para los pesos por cada diámetro R

En cuanto a las métricas definidas para los pesos, obtenemos resultados esperables, en el caso del Modelo B al ser más chico el paso que da en cada iteración el algoritmo, obtenemos que tanto la trayectoria T como la dispersión de los vectores finales es menor para todo valor de R y además el parámetro de W_e para el Modelo B es comparable al de W_b, es decir la distancia al $\bar{W}_I$ interpolador es semejante al comienzo y al final de las iteraciones.

R	W_b	W_e Mod A	W_e Mod B
0.9	6.9188	52.2131	6.8560
0.5	3.8438	13.0464	3.8119
0.1	0.7688	22.5884	0.7437
0.05	0.3844	15.3187	0.3666
0.01	0.0769	17.5881	0.0719
0.005	0.0384	10.3855	0.0357
0.001	0.0077	13.0030	0.0071
0.0005	0.0038	16.6428	0.0036
0.0001	0.0008	22.1406	0.0007
0.00005	0.0004	13.9061	0.0004
0.00001	0.0001	12.6756	0.0001

Tabla 6.5: Tabla de Métricas para los pesos por cada diámetro R

Veamos los gráficos comparativos de las predicciones resultante del Modelo A y el Modelo B. Los vectores de pesos iniciales fueron los mismos en todos los casos.

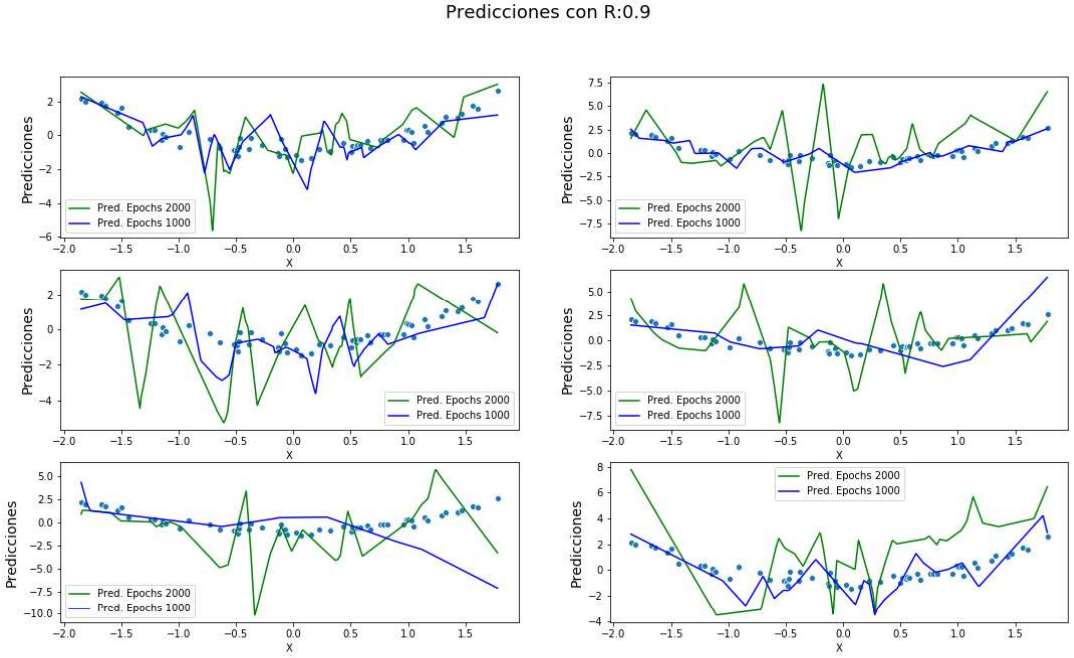

Figura 6.19: Predicciones obtenidos con R = 0.9

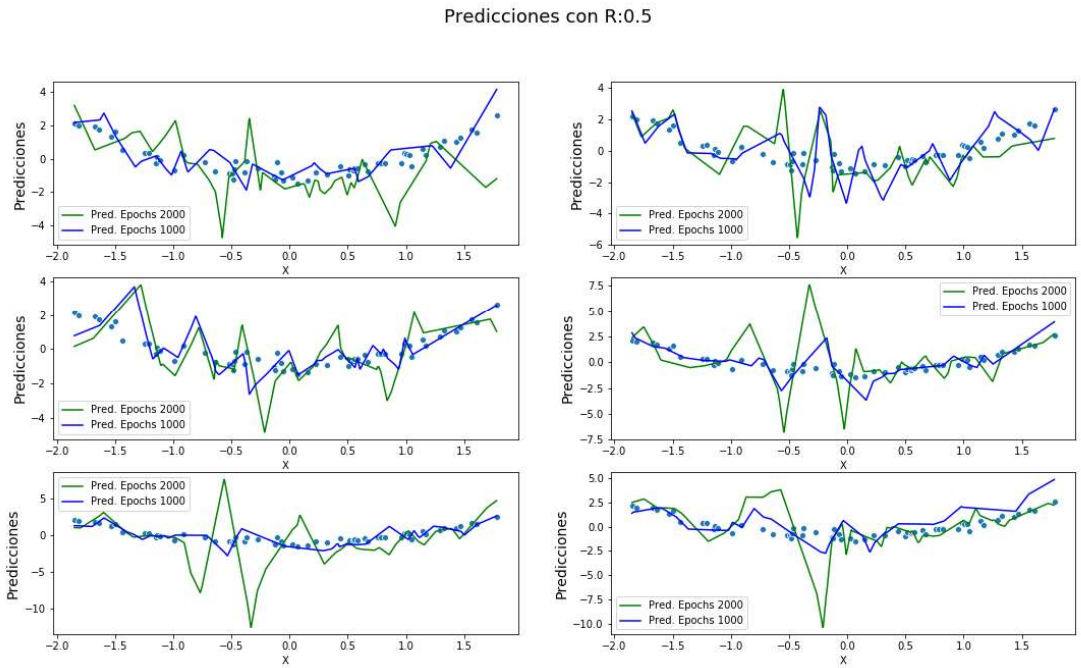

FIGURA 6.20: Predicciones obtenidos con R = 0.5

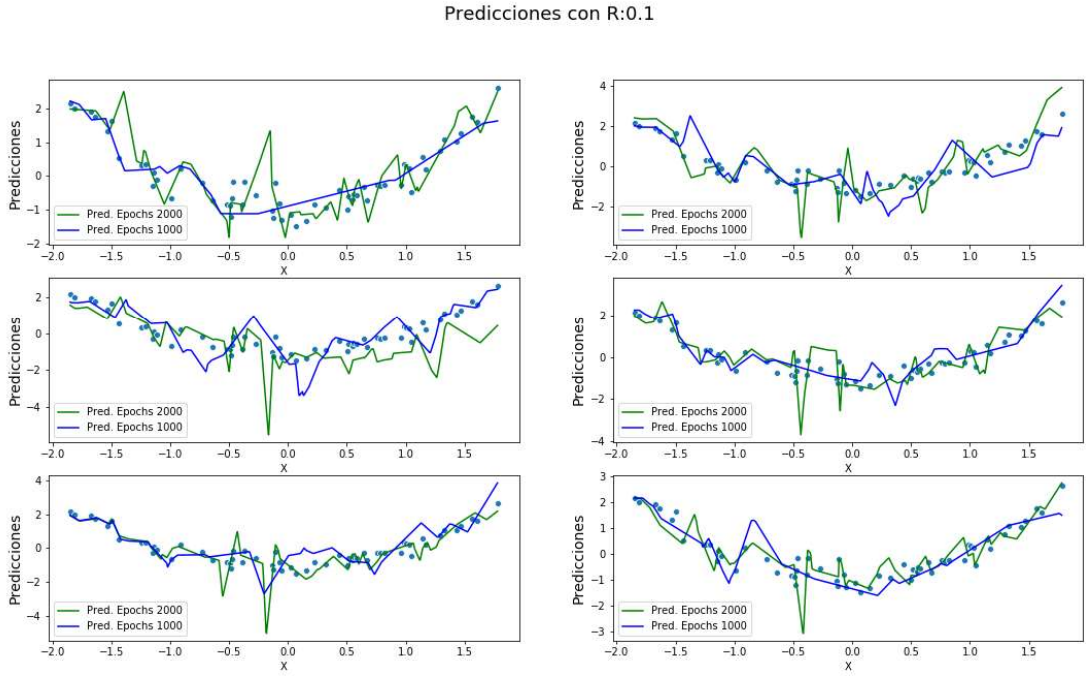

FIGURA 6.21: Predicciones obtenidos con R = 0.1

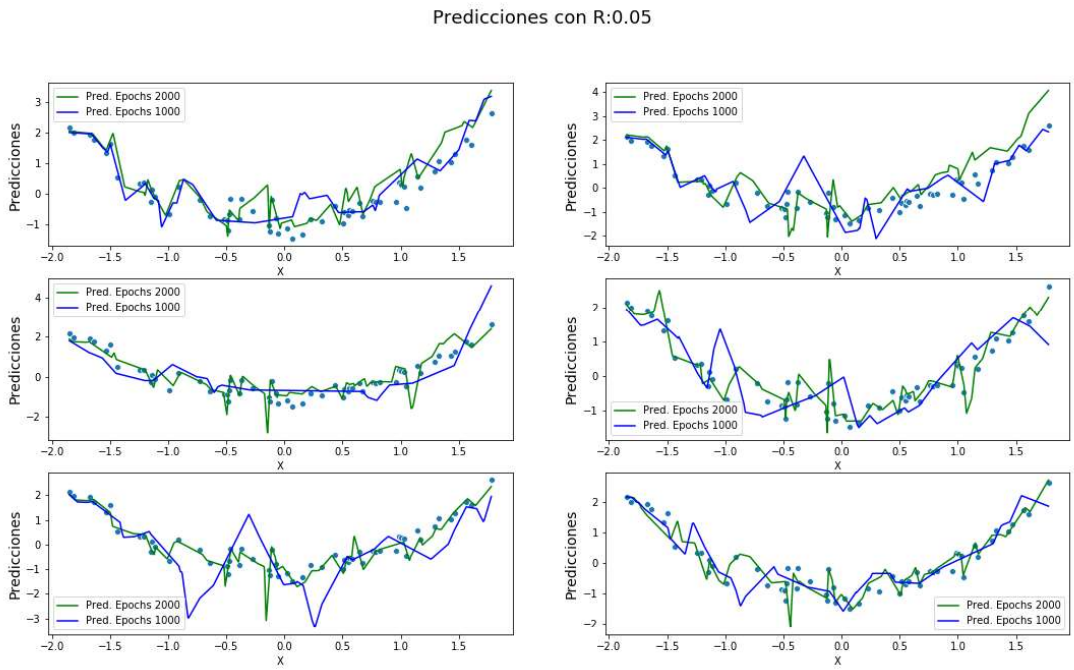

FIGURA 6.22: Predicciones obtenidos con R = 0.05

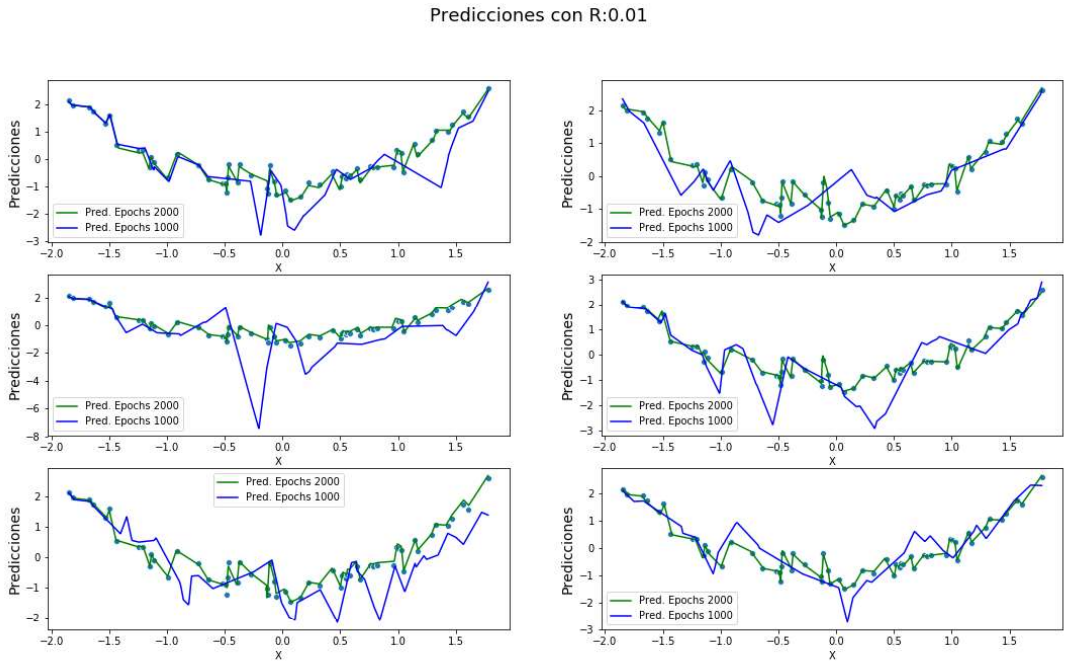

FIGURA 6.23: Predicciones obtenidos con R = 0.01

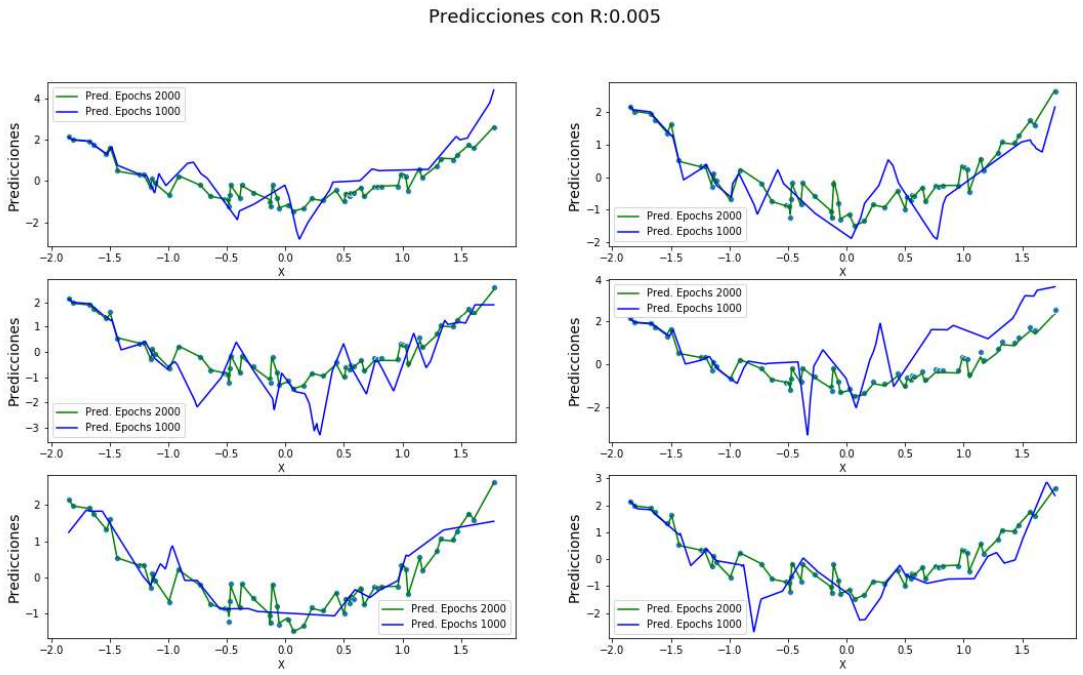

FIGURA 6.24: Predicciones obtenidos con R = 0.005

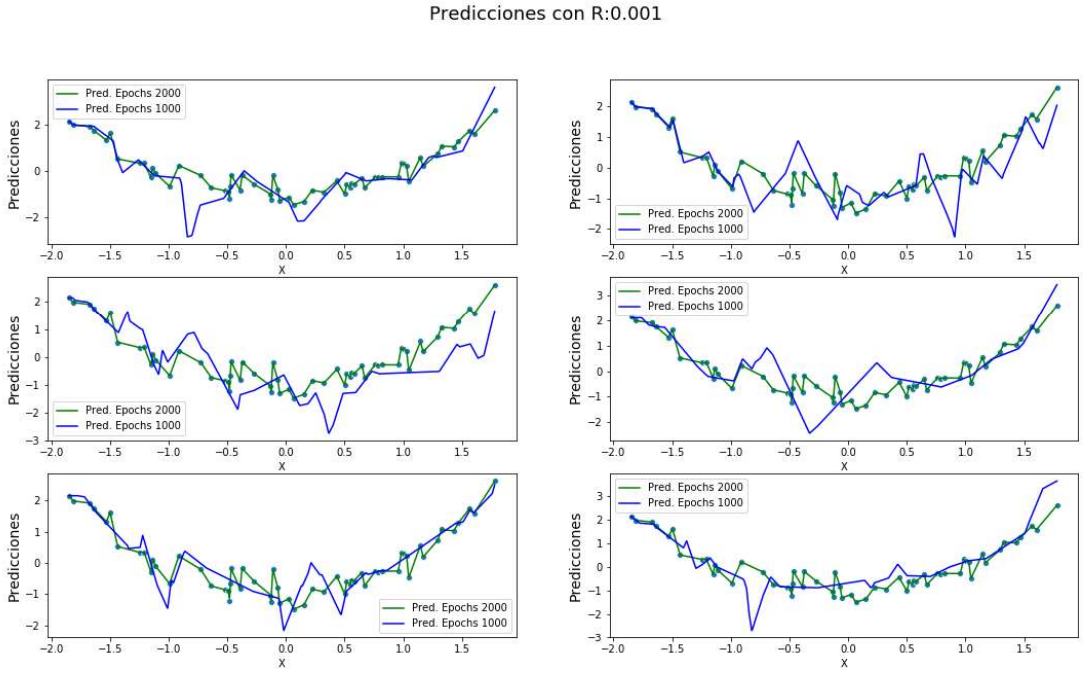

FIGURA 6.25: Predicciones obtenidos con R = 0.001

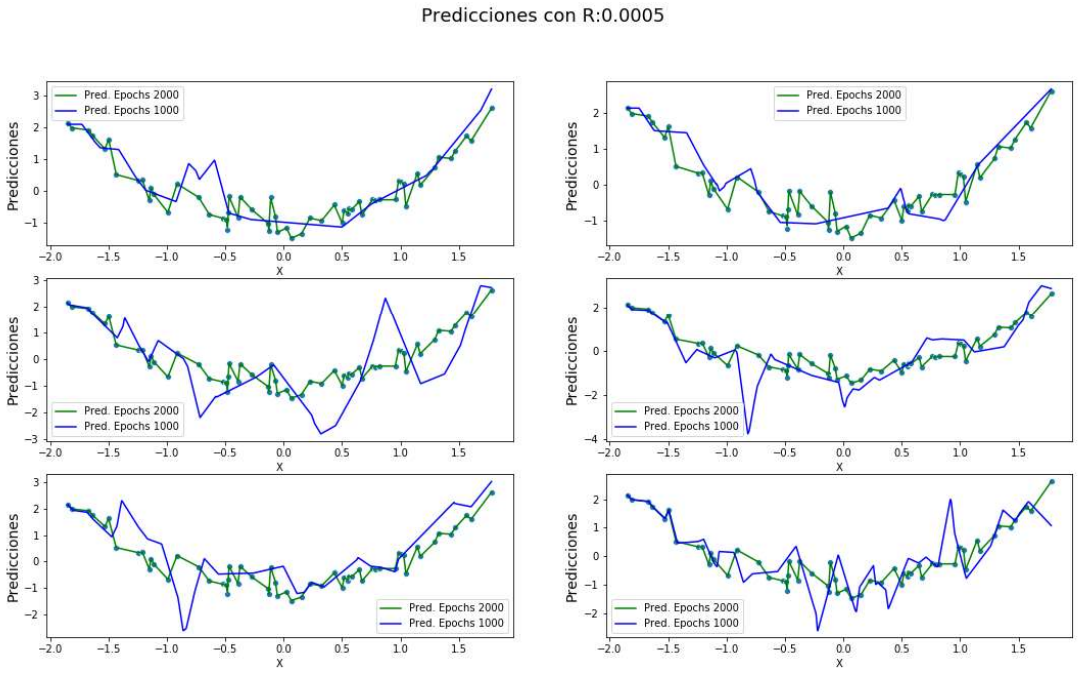

FIGURA 6.26: Predicciones obtenidos con R = 0.0005

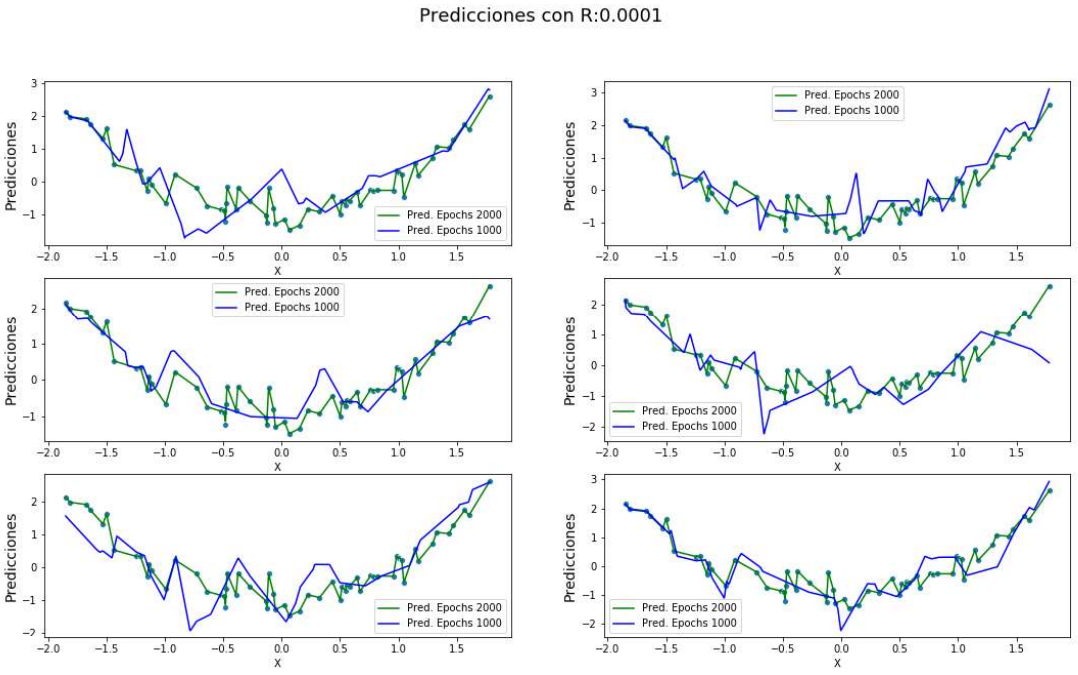

FIGURA 6.27: Predicciones obtenidos con R = 0.0001

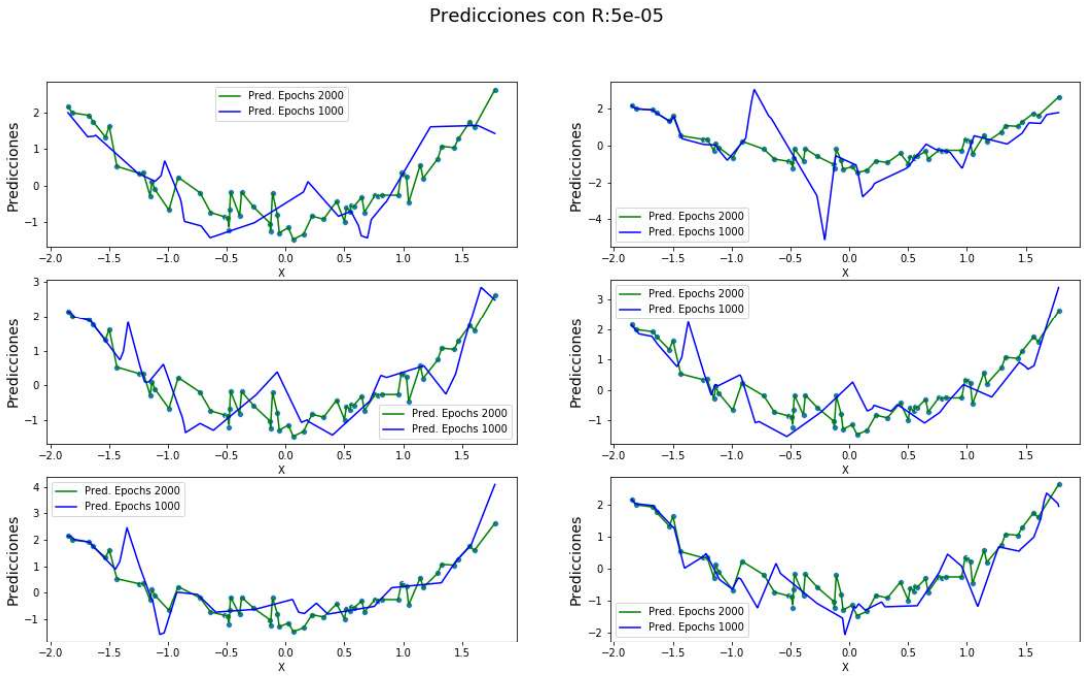

FIGURA 6.28: Predicciones obtenidos con R = 0.00005

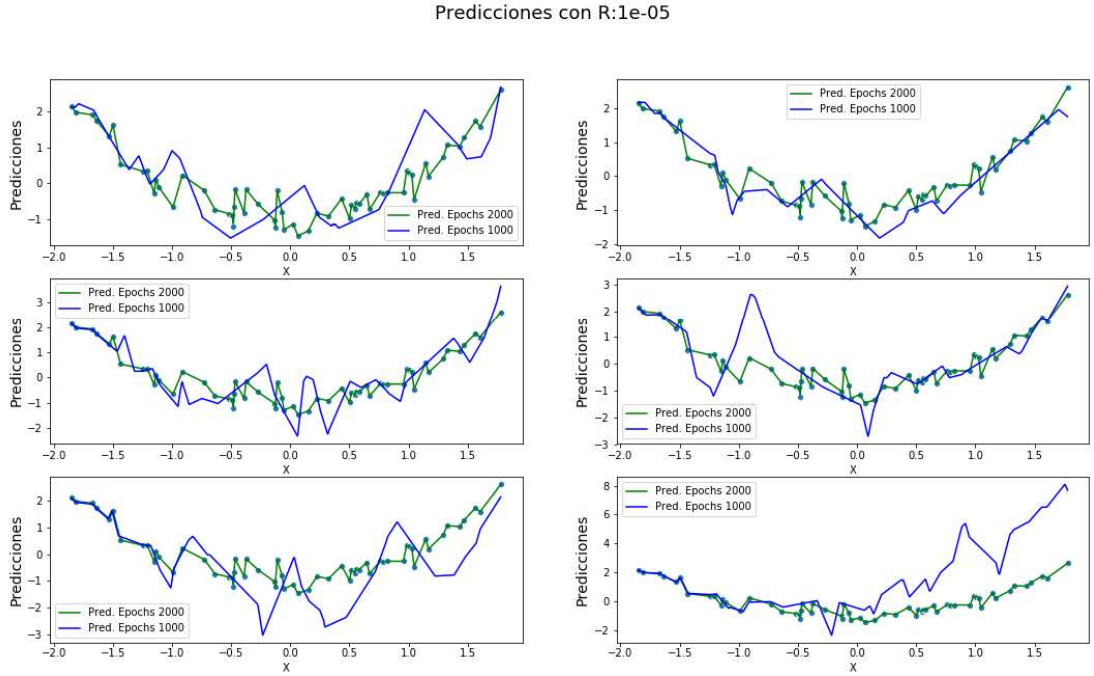

FIGURA 6.29: Predicciones obtenidos con R = 0.00001

De forma consecuente a la tabla 6.3, para los valores más grandes de R, el Modelo A genera mejores predicciones que el Modelo B, mientras que para los valores más chicos las predicciones obtenidas por el Modelo B son 'casi' la interpolación.

Con este análisis podemos ver la importancia de los hiperparámetros del algoritmo optimizador, el compromiso que hay entre el paso que se da en cada iteración (*learning rate*) y la cantidad de iteraciones (*epochs*). Si el *learning rate* es muy chico, por más iteraciones que se realicen el vector no logra alejarse mucho de su posición inicial y en este caso el resultado final va a depender de la posición inicial. Es importante rescatar que en este caso interpolador, para que el algoritmo alcance al mínimo global se tuvo que forzar tanto el punto inicial (que sea lo suficientemente cerca del mínimo) como definir un paso infinitamente chico y darle una gran cantidad de iteraciones para poder alcanzar el mínimo, si alguna de estas condiciones no se cumplían entonces el resultado iba a ser una aproximación de los datos y el algoritmo iba a alcanzar un mínimo local y no global.

Capítulo 7

Conclusiones

En primer lugar hemos mostrado evidencia empírica que en contextos simples de regresión los ajustes mediante GD de Redes Neuronales artificiales son por lo general parsimoniosos. Pese a su carácter de aproximadores universales, y a su capacidad teórica de hallar mínimos globales, la interpolación de los datos es muy difícil de conseguir.

En segundo lugar, hemos desarrollado un método de cálculo que devuelve los pesos de una Red formada por N-1 neuronas que interpola N datos, y vimos además también como extender el cálculo con el agregado de más neuronas, manteniendo la situación de interpolación de los datos.

En tercer lugar hemos logrado generar situaciones, utilizando este método de cálculo que devuelve los pesos de una Red interpoladora, en las que el algoritmo GD halla el mínimo global. Precisamente vimos que para llegar al mínimo global, el algoritmo optimizador GD debe incializarse en un vector en el espacio de pesos que se encuentre a una distancia extremadamente pequeña del mínimo global, además de que se debe fijar el paso del algoritmo GD en valores extremadamente chicos. Seguidamente, hemos mostrado que en la práctica estas circunstancias de convergencia al mínimo global no se alcanzan, llegándose generalmente a ajustes parsimoniosos de los datos.

La conclusión final de este trabajo podría resumirse en la idea que la 'magia' de las redes neuronales artificiales podría recaer en la no efectivización de su capacidad expresiva total y de su potencialidad optimizadora, sino en la de encontrar implícitamente, eventualmente sin la necesidad de mecanismos regularizadores explícitos, situaciones intermedias en las que el proceso subyacente queda bien representado, generando de esta manera un enorme potencial de generalización.

Bibliografía

Baldi, P. and K. Hornik (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*@(1), 53–58.

Baldi, P. and Z. Lu (2012). Complex-valued autoencoders. *Neural Networks 33, 136–147.*

Choromanska, A., M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun (2015). The loss surfaces of multilayer networks. In Artificial Intelligence and Statistics*, pp. 192–204.*

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems*@(4), 303–314.*

Du, S. S., J. D. Lee, H. Li, L. Wang, and X. Zhai (2018). Gradient descent finds global minima of deep neural networks. arXiv preprint arXiv:1811.03804.

Feizi, S., H. Javadi, J. Zhang, and D. Tse (2017). Porcupine neural networks:(almost) all local optima are global. arXiv preprint arXiv:1710.02196.

Freeman, C. D. and J. Bruna (2016). Topology and geometry of half-rectified network optimization. arXiv preprint arXiv:1611.01540.

Gori, M. and A. Tesi (1992). On the problem of local minima in backpropagation. IEEE Transactions on Pattern Analysis & Machine Intelligence *(1), 76–86.*

Hanin, B. (2017). Universal function approximation by deep neural nets with bounded width and relu activations. arXiv preprint arXiv:1708.02691.

Hanin, B. and M. Sellke (2017). Approximating continuous functions by relu nets of minimal width. arXiv preprint arXiv:1710.11278.

Hinton, G., L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury, et al. (2012). Deep neural networks for acoustic modeling in speech recognition. IEEE Signal processing magazine 29.

Hinton, G. E. and S. T. Roweis (2003). Stochastic neighbor embedding. In *Advances in neural information processing systems*, pp. 857–864.

Hornik, K., M. Stinchcombe, and H. White (1989, July). Multilayer feedforward networks are universal approximators. *Neural Netw.*@(5), 359–366.

Kawaguchi, K. (2016). Deep learning without poor local minima. In *Advances in neural information processing systems*, pp. 586–594.

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105.

Lee, J. D., M. Simchowitz, M. I. Jordan, and B. Recht (2016). Gradient descent converges to minimizers. *arXiv preprint arXiv:1602.04915*.

Li, Y., J. Yosinski, J. Clune, H. Lipson, and J. E. Hopcroft (2016). Convergent learning: Do different neural networks learn the same representations? In *Iclr*.

Lu, H. and K. Kawaguchi (2017). Depth creates no bad local minima. *arXiv preprint arXiv:1702.08580*.

Lu, Z., H. Pu, F. Wang, Z. Hu, and L. Wang (2017). The expressive power of neural networks: A view from the width. In *Advances in Neural Information Processing Systems*, pp. 6231–6239.

Maaten, L. v. d. and G. Hinton (2008). Visualizing data using t-sne. *Journal of machine learning research*@(Nov), 2579–2605.

Maennel, H., O. Bousquet, and S. Gelly (2018). Gradient descent quantizes relu network features. *arXiv preprint arXiv:1803.08367*.

Nguyen, Q. and M. Hein (2017). The loss surface of deep and wide neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2603–2612. JMLR. org.

Raghu, M., B. Poole, J. Kleinberg, S. Ganguli, and J. S. Dickstein (2017). On the expressive power of deep neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2847–2854. JMLR. org.

Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*@(7587), 484.

Soudry, D. and Y. Carmon (2016). No bad local minima: Data independent training error guarantees for multilayer neural networks. *arXiv preprint arXiv:1605.08361*.

Telgarsky, M. (2015). Representation benefits of deep feedforward networks. *arXiv preprint arXiv:1509.08101*.

Telgarsky, M. (2016). Benefits of depth in neural networks. *arXiv preprint arXiv:1602.04485*.

Yu, X.-H. and G.-A. Chen (1995). On the local minima free condition of backpropagation learning. *IEEE Transactions on Neural Networks*@(5), 1300–1303.

Yun, C., S. Sra, and A. Jadbabaie (2017). Global optimality conditions for deep neural networks. *arXiv preprint arXiv:1707.02444*.

Zhang, C., S. Bengio, M. Hardt, B. Recht, and O. Vinyals (2016). Understanding deep learning requires rethinking generalization.

Zhou, Y. and Y. Liang (2017). Critical points of neural networks: Analytical forms and landscape properties. *arXiv preprint arXiv:1710.11205*.

Zou, D., Y. Cao, D. Zhou, and Q. Gu (2018). Stochastic gradient descent optimizes over-parameterized deep relu networks. *arXiv preprint arXiv:1811.08888*.