REDES NEURONALES
Trabajo Final
Red Feed Forward Multicapas Análisis de extrapolación sobre Funciones Continuas y Parcialmente Continuas

Prof: Francisco A. Tamarit
Prof: Sergio Cannas
Autor: Eric N. Jurio

23 de diciembre de 2013

Resumen

Las redes neuronales feed forward (feed forward neural networks, desde ahora FFNN) se usan en muchos casos para aproximación de funciones. Para entrenarlas se les brinda cierto conjunto de datos, los cuales luego interpolaran. Por lo general el objetivo es trabajar con la FFNN aproximando a la función dentro de cierto radio entre los puntos de entrenamiento, pero uno también podría intentar trabajar con la FFNN fuera de el radio que cubren los puntos de entrenamiento (pasar de interpolar a extrapolar). La calidad con que la FFNN aproxime a la función objetivo dependerá de muchos factores (cantidad de puntos de entrenamiento, estructura de la FFNN, función de activación de cada capa de la FFNN, etc). Muchos de estos factores quedan a criterio de cada desarrollador/investigador, dependiendo del problema y casi siempre trabajando por prueba y error. En la literatura no existe un criterio estándar aceptado, o método para la selección de todos estos parámetros. Sobre todo en cuanto a estructura y funciones de activación. Por lo general se eligen una capa o dos y con funciones de activación $tanh(x)$ (tangente hiperbólica) o la función lineal.

En este trabajo nos centramos en el análisis de distintas variantes de estructuras y funciones de activación, haciendo inca pie en como se comportan al momento de la extrapolación. Analizamos tres escenarios distintos de funciones continuas o parcialmente continuas y en cada uno comparamos tres redes, con estructura tradicional, con función de activación cambiada y una red cuya estructura crece a demanda durante el entrenamiento (construcción adaptativa).

Introducción

Las redes neuronales perceptron multicapas o FFNN han sido ampliamente utilizadas debido a su simplicidad y buenos resultados, aunque en ocasiones fallan en proveer una adecuada solución, debido a una mala arquitectura, insuficiente número de neuronas, insuficientes número de ciclos de entrenamiento, mala elección de la función de activación o cantidad insuficiente de información para el entrenamiento respecto de lo que se desea aproximar o generalizar. El desempeño del entrenamiento de una red neuronal depende del algoritmo de aprendizaje utilizado, del número de capas ocultas, del número de neuronas en cada capa oculta, de la conectividad o arquitectura y también del tipo de función de activación que se elija para cada neurona.

Motivación: ¿Por qué es necesario elegir bien las funciones de activación?

Elegir bien la estructura de la red es importante ya que determina que tan bien asimila los conceptos que tratamos de entrenar en la FFNN. Una red pequeña (con pocas neuronas en su capa oculta) necesita demasiado entrenamiento y quizás sobre entrenamiento para aprender algunos conceptos y probablemente no generalice bien. Una red con pocas capas ocultas, puede no tener capacidad suficiente para aprender y generalizar ciertas funciones.

Así también es importante saber cuando es necesario usar un tipo de función de activación y cuando otro tipo.

Digamos que se quiere emular cierto comportamiento de algún sistema y se sabe que la función que describe ese comportamiento tiene ciertas características, pero se desconoce la función en si misma. (por ejemplo se que la función tiene una fuerte componente sinusoidal, o es producto de otras funciones, o que es cuadrática y creciente, o simplemente no esta acotada y es un polinomio de grado mayor a 1...). Además el conjunto de datos actual es limitado y describe una parte del comportamiento, pero se sabe que a futuro el sistema se comportara bajo las mismas reglas pero en otro rango de valores para los parámetros de entrada. Esto implicaría que la red trabaje extrapolando datos, por lo que (en este tipo de casos) es muy importante a la hora de construir la FFNN, no solo que interpole bien el conjunto de entrenamiento, sino también que a nivel estructura capture bien la función que intenta imitar.

Por ejemplo, en el caso de que la función objetivo fuere de ramas ascendentes y solo este acotada por debajo, no seria una buena idea tratar de aproximarla con una FFNN de una capa oculta $tanh(x)$ y salida lineal, ya que al tratar de extrapolar no podría imitar las ramas ascendentes. Entonces supongo que puedo obtener un mejor resultado entrenando una red neuronal que tenga cierta función de activación diferente en algunas neuronas.

Objetivos

En este informe analizaremos a las FFNN aplicadas a funciones continuas. Para ello se plantean los siguientes objetivos:

1. Aproximaremos las funciones $\frac{1}{x}$ y $\log(x)$, trataremos de extrapolar y compararemos con alternativas.

2. Aproximaremos la función $f(x, y) = x * y$ con FFNN. Analizaremos distintas opciones y tomaremos ejemplos de extrapolación.

3. Analizaremos distintas opciones para aproximar la función $sinc(x)*sinc(y) = \frac{sin(x)}{x} * \frac{sin(y)}{y}$ y analizaremos la extrapolación.

En cada uno de estos puntos trabajaremos con FFNN construidas de distintas maneras:

- FFNN con elección de parámetros y estructura de manera tradicional (1 capa oculta con función de activación $tanh(x)$ y capa de salida con función de activación lineal)
- FFNN con función de activación ad-hoc (particular para el caso) en la capa oculta, e incluso mas de un tipo de función de activación por capa.
- FFNN con crecimiento adaptativo. Donde la propia red alcanza la mejor estructura dependiendo del conjunto de entrenamiento y pocos parámetro mas.

Campo de conocimiento

Generalidades sobre las redes neuronales Feed Forward

La red neuronal feed forward fue el primer tipo y posiblemente el más simple de red neuronal artificial ideado. En esta red, la información se mueve en una sola dirección (hacia adelante), a partir de los nodos de entrada, a través de los nodos ocultos (si los hay) terminando en los nodos de salida. Como característica principal destacamos que es un red sin bucles de flujo de información.

Están inspiradas en redes neuronales biológicas (en particular, en el sistema nervioso de los animales).

Este tipo de redes son aplicada para aprendizaje de patrones, mapeo de funciones (control) y clasificación, principalmente. Una de las aplicaciones que tiene es para el aprendizaje de funciones con su dominio e imagen en espacios n-dimensionales de tipo binario, por eso estas redes son también conocidas como redes perceptron. Pero el modelo no limita solo a casos binarios, ya que se puede variar la función de activación de las neuronas (neuronas binarias, lineales, continuas derivables no lineales, etc).

En particular en este trabajo planteamos la hipótesis de que la correcta selección de la función de activación de las neurona de la red juega un papel importante en los resultados del entrenamiento y de la aplicación de la red, sobre todo si necesitamos hacer extrapolación de datos.

Alcance del trabajo y Marco general

En este trabajo solo se plantea un enfoque o manera en la que se pueden usar las FFNN en donde se pueden producir problemas (al extrapolar). En este informe se presentan resultados a modo comparativo de distintas aproximaciones por distintas estrategias.

No se pretende probar, ni afirmar nada. Solo es un trabajo de investigación exploratorio donde exponemos e interpretamos resultados.

Sobre la implementación

He desarrollado mi propia librería open source para el modelo perceptron (entre otros modelos que continuo desarrollando).

No voy a entrar en detalles sobre la implementación. Algunas características implementadas son:

- Implementación de redes feed forward simple y multicapas. (definición de numero de capas, tipo de neurona en la capa y numero de neuronas por capa al momento de construcción de la red)
- Aprendizaje de conjuntos de entrenamientos en modo batch u online.
- Aprendizaje con momento. (momento por defecto de 0,5 o seleccionable en $[0{,}0; 1{,}0]$)
- Parámetro de adaptación auto-regulado en un rango preseleccionado (rango por defecto $[0{,}02; 0{,}05]$; rango ajustable en $[0{,}0; 1{,}0]$)
- Implementación de crecimiento adaptativo de la red feed forward durante el entrenamiento.

Para mas información de las características y del proceso de desarrollo ver en la pagina del proyecto:

El proyecto (en general) se encuentra en https://bitbucket.org/ericnjurio/neural-networks

La implementación en particular para este trabajo esta en http://code.google.com/p/redes-neuronales/ (utiliza librerías del proyecto recién mencionado)

Aproximación de funciones continuas

Variantes de Redes Feed Forward Multicapas

En todos los escenarios que plantearemos a continuación, trataremos de aproximar funciones en 2 dimensiones y en 3 dimensiones, por lo que las redes neuronales tendrán 1 o 2 entradas y solo una salida.

En cada uno de los escenarios planteados en el trabajo compararemos el comportamiento de tres redes neuronales.

1. En el primer caso veremos FFNN con construcción típicas con una capa oculta de función de activación $tanh(x)$ y capa de salida lineal. Para cada escenario usaremos distintas cantidades de neuronas según sea necesario.

2. En el segundo caso veremos FFNN con una capa oculta pero distintas funciones de activación según sea necesario, inclusive con capas ocultas mixtas (una capa oculta con mas de un tipo de función de activación) y capa de salida lineal.

3. En el tercer caso veremos FFNN con crecimiento adaptativo la cual puede variar en cantidad de neuronas por capas y en cantidad de capas según sea necesario, acorde al error FVU (fraction of variance unexplained), el cual es indicativo de si la red esta lo suficientemente adaptada a los datos.

$$FVU = \frac{\sum_{j=1}^{P} \left(\hat{g}\left(x_j\right) - g\left(x_j\right)\right)^2}{\sum_{j=1}^{P} \left(g\left(x_j\right) - \overline{g}\right)^2}$$

Donde $g\left(\cdot\right)$ es la función a aproximar por FFNN, $\hat{g}\left(\cdot\right)$ es la estimación de $g\left(\cdot\right)$ dada por FFNN y $\overline{g}$ es la media de $g\left(\cdot\right)$, x_j es una muestra (la j-esima) del conjunto de puntos de entrenamiento y P es el total de muestras en el conjunto de entrenamiento. Para mas detalles sobre el algoritmo ver las referencias.

Dado que este tipo de red que crece, en algún punto cuando agrega nuevas capas, duplica la ultima capa oculta o en el caso de no tener capa oculta, genera una basado en la capa de salida, clonándola y por tener la particularidad de que nuestra capa de salida siempre sera lineal; en consecuencia tenemos que en cada nueva capa que se agregue tendremos al menos una neurona de función de activación de tipo lineal. Luego las salidas tendrán siempre una componente lineal directa de las entradas (mas o menos ponderada dependiendo el caso).

Análisis de funciones continuas en Interpolación y Extrapolación

Para todos los casos usamos el algoritmo de aprendizaje backpropagation (on-line) en FFNN con capa oculta de M neuronas para aproximar funciones continuas de una variable $f(x)$ o dos variables $f(x, y)$. Como función de activación para las neuronas de la capa oculta usamos la función $\tanh(x)$ (en la mayoría de los casos) y para la neurona de la capa de salida usamos la función lineal: $OutNet = \sum_{i=1}^{M}(outHide_i * w_i)$ donde OutNet es la salida de la neurona de salida (lineal), $outHide_i$ son las salidas de las neuronas de la capa oculta y w_i son los pesos sinápticos de entrada a la neurona lineal.

Escenario 1: Aproximación de $\frac{1}{X}$ y $\ln(x)$

Entrenamos sobre p puntos de la función tomados aleatoriamente en un intervalo finito de la variable independiente x. Imponiendo un limite ϵ de tolerancia de error de aprendizaje por patrón. Aproximamos las siguientes funciones en el intervalo $x \in [1; 5]$:

- $f(x) = \frac{1}{X}$ con $M = 4$, $\epsilon = 0{,}01$ para $p = 5; 10; 20$.
- $f(x) = \ln(x)$ con $M = 6$, $\epsilon = 0{,}02$ para $p = 5; 10; 20$.

Donde M es la cantidad de neuronas en la capa oculta.

Para cada función ($\frac{1}{X}$ y $\ln(x)$) realizaremos aproximaciones con los tipos de redes neuronales plateados anteriormente. Para nuestro caso de "FFNN con función de activación no convencional" analizaremos cada función con otras dos funciones de activación: $f_{act}(x) = \frac{1}{(x^2+0{,}005)}$ y $f_{act}(x) = ln(x^2 + 0{,}05)$

Dado que en general, las aproximaciones son buenas, resulta difícil comparar a simple vista las funciones y las aproximaciones, por lo que decidimos graficar la diferencia $g(x_j) - \hat{g}(x_j)$ (erro de aproximación). Donde $g(\cdot)$ es la función a aproximar por FFNN y $\hat{g}(\cdot)$ es la estimación de $g(\cdot)$ dada por FFNN

A continuación podemos ver las gráficas simultáneas de el error de aproximación para distintas cantidades de puntos de entrenamientos.

Aproximación de $f(x) = \frac{1}{X}$

A continuación vemos la Figura 0.1, tres curvas de error de aproximación superpuestas. (3 aproximaciones dadas por FFNN típica con una capa oculta de 4 neuronas de función de activación $tanh(x)$ y capa de salida lineal)

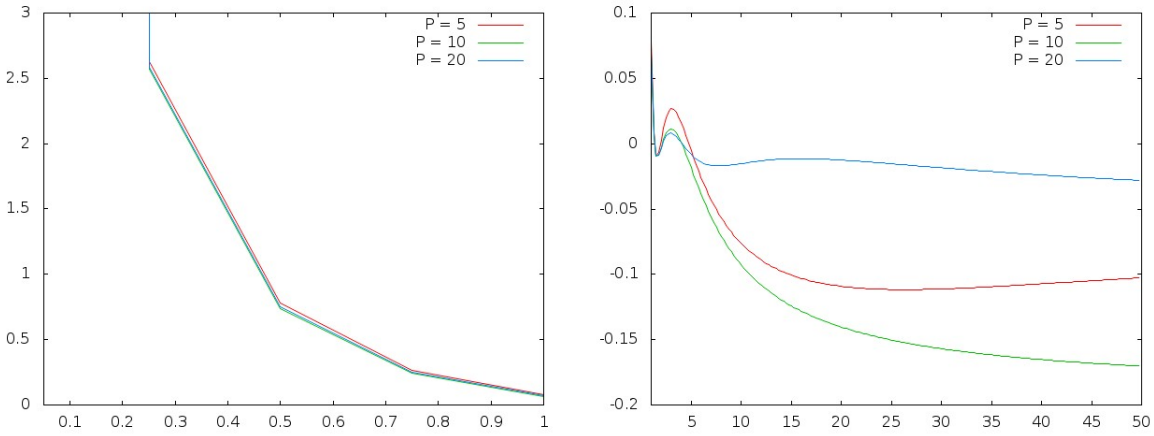

Figura 0.1: Error de aproximación a $f(x) = \frac{1}{X}$ por una red típica de 4 neuronas de capa oculta y salida lineal, entrenadas con 5, 10 y 20 puntos.

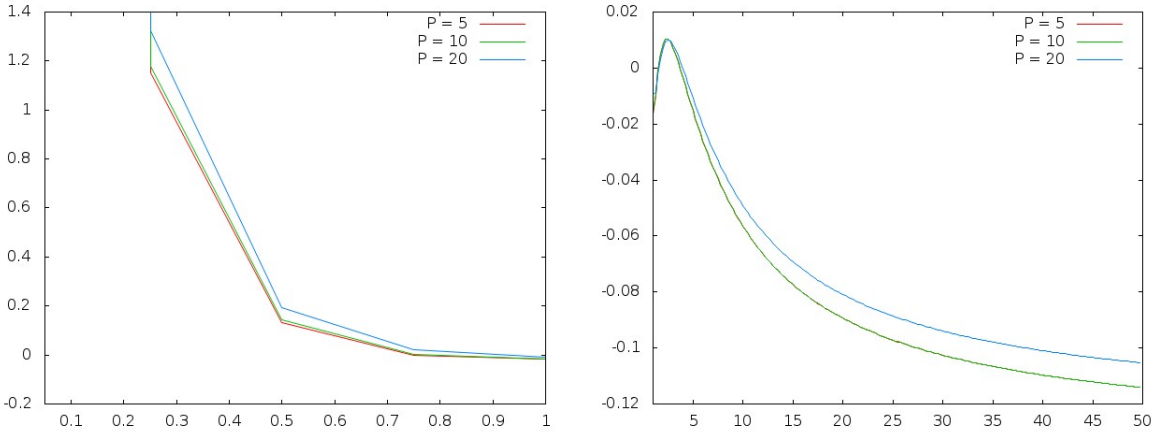

Figura 0.2: Error de aproximación a $f(x) = \frac{1}{X}$ por una red con una capa oculta de 4 neuronas con función de activación $f_{act}(x) = \frac{1}{(x^2+0{,}005)}$ y salida lineal, entrenadas con 5, 10 y 20 puntos.

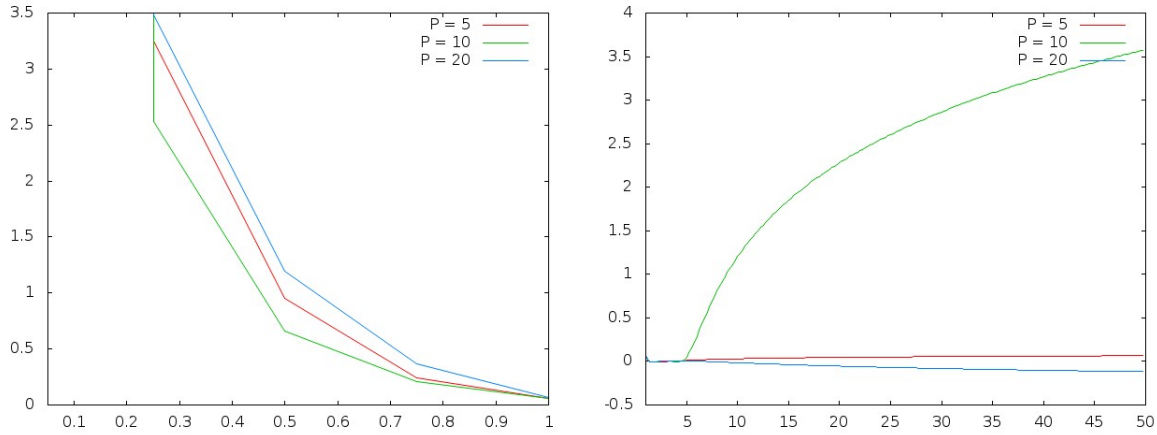

Figura 0.3: Error de aproximación a $f(x) = \frac{1}{X}$ por una red con una capa oculta de 4 neuronas con función de activación $f_{act}(x) = ln(x^2 + 0{,}05)$ y salida lineal, entrenadas con 5, 10 y 20 puntos.

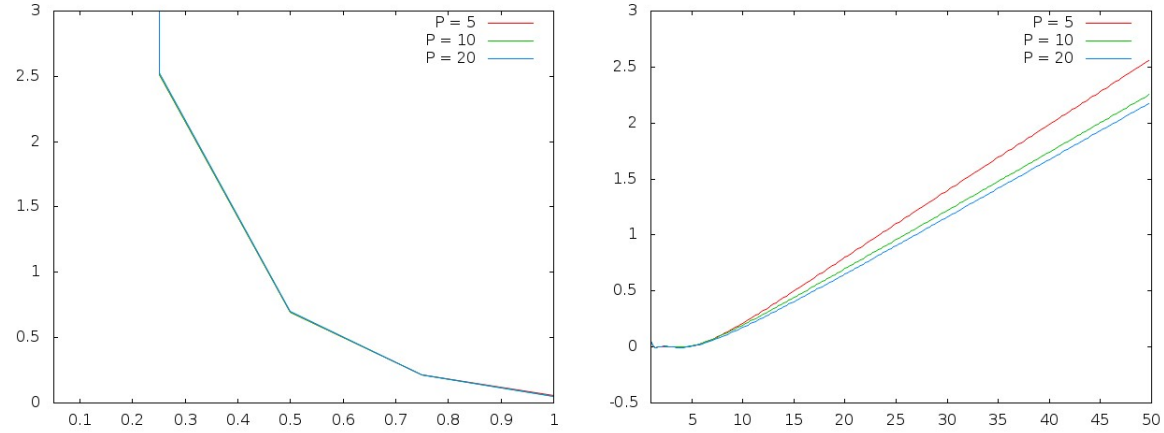

Figura 0.4: Error de aproximación a $f(x) = \frac{1}{X}$ por una red obtenida de un algoritmo de construcción, con capa/s oculta/s con función de activación $f_{act}(x) = tanh(x)$ combinadas con neuronas lineales y salida lineal, entrenadas con 5, 10 y 20 puntos.

Podemos ver que las FFNN con capa oculta $f_{act}(x) = tanh(x)$ sola y la FFNN con capa oculta $f_{act}(x) = \frac{1}{(x^2+0{,}005)}$ aproximan mejor a la función $f(x) = \frac{1}{X}$ (Figura 0.1 y 0.2); en contraposición de las FFNN con capa oculta $f_{act}(x) = ln(x^2 + 0{,}05)$ y con capa oculta mixta $f_{act}(x) = \{tanh(x), lineal\}$ (Figura 0.3 y 0.4). Podemos suponer que las primeras aproximan mejor en el

intervalo [1,0; 50,0] ya que son acotadas y comparten que $\lim_{x\to\infty} g(x) = C$ con C finito. Las otras no tienen cotas ($f_{act}(x) = ln(x^2 + 0{,}05)$ solo es acotada por debajo y no tiene limite finito cuando el dominio tiende a ∞ o $-\infty$), por lo que al extrapolar, el error es cada vez mayor.

Ninguna extrapola bien en el intervalo [0,0; 1,0] (el error aumenta cuando nos acercamos a 0), podemos suponer que esto se debe a que la función $f(x) = \frac{1}{X}$ tiende a infinito cuando x tiende a cero y ninguna de las funciones de activación puede aproximar eso, ya que el algoritmo de entrenamiento que usamos (backpropagation) tiene como requerimiento que la función de activación debe ser derivable en todo $\mathbb{R}$. Analizando un poco podemos decir que, las FFNN a las cuales entrenemos con backpropagation, podremos usarlas para aproximar funciones discontinuas hasta cierto punto, determinado por la cercanía de los puntos de entrenamiento a el punto de discontinuidad. Como comentario adicional, podemos decir que si quisiéremos utilizar funciones de activación discontinuas para aproximar funciones discontinuas, podríamos intentarlo, usando otros algoritmos de entrenamientos (como por ejemplo algoritmos evolutivos o genéticos sobre FFNN)

Aproximación de $f(x) = \ln(x)$

A continuación vemos la Figura 0.5, tres curvas de error de aproximación superpuestas. (3 aproximaciones dadas por FFNN típica con una capa oculta de 6 neuronas de función de activación $tanh(x)$ y capa de salida lineal)

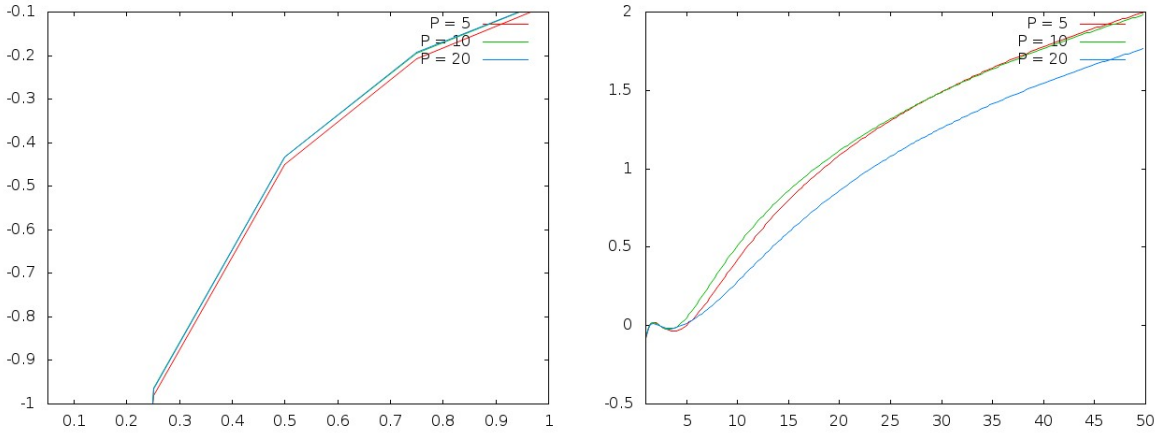

Figura 0.5: Error de aproximación a $f(x) = \ln(x)$ por una red típica de 6 neuronas de capa oculta y salida lineal, entrenadas con 5, 10 y 20 puntos.

En la Figura 0.5 podemos ver que la aproximación es buena cerca de los puntos entrenados y el error es más grande al alejarse. De echo al alejarse de los

puntos de entrenamiento el error se puede describir por $\ln(x)$, que es la propia función objetivo. Esto se debe a que $tanh(x)$ esta acotada por arriba y por abajo y la función objetivo no.

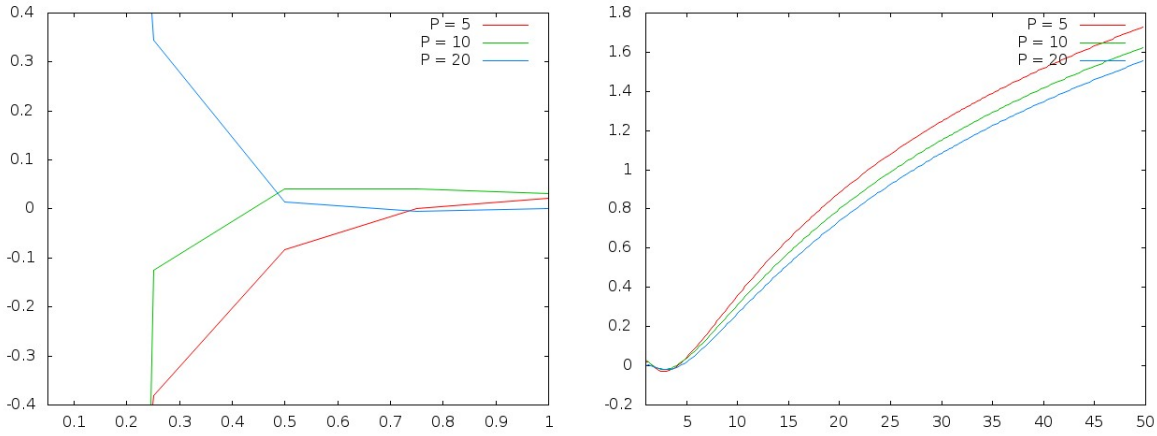

Figura 0.6: Error de aproximación a $f(x) = \ln(x)$ por una red con una capa oculta de 6 neuronas con función de activación $f_{act}(x) = \frac{1}{(x^2+0{,}005)}$ y salida lineal, entrenadas con 5, 10 y 20 puntos.

En la figura 0.6 observamos lo mismo que en la figura 0.5. Aquí además podemos notar que, a mayor cantidad de puntos de entrenamiento, mejor es la interpolación y la extrapolación.

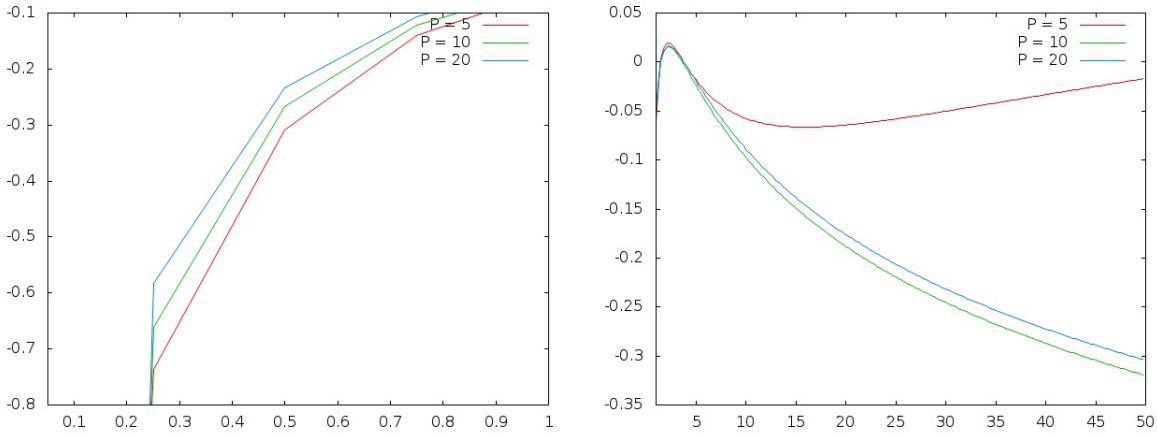

Figura 0.7: Error de aproximación a $f(x) = \ln(x)$ por una red con una capa oculta de 6 neuronas con función de activación $f_{act}(x) = ln(x^2 + 0{,}05)$ y salida lineal, entrenadas con 5, 10 y 20 puntos.

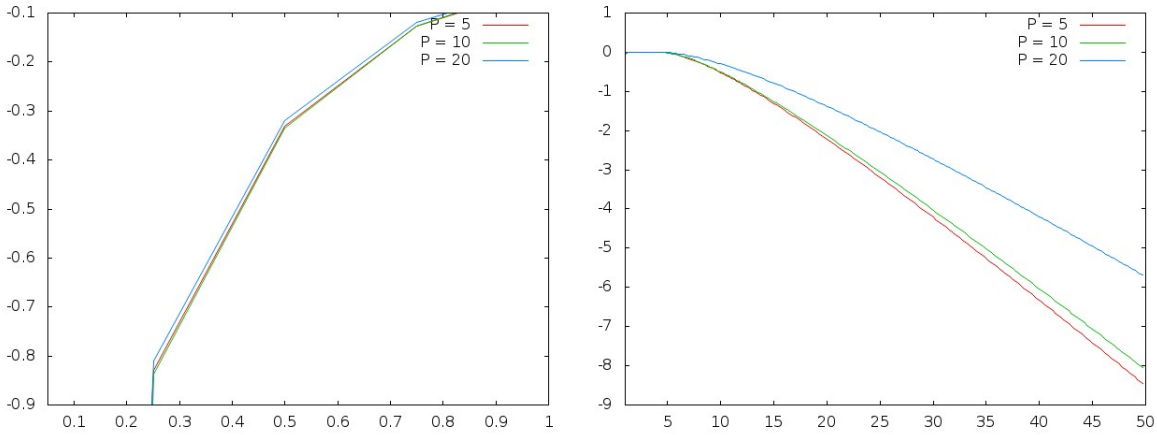

Figura 0.8: Error de aproximación a $f(x) = \ln(x)$ por una red obtenida de un algoritmo de construcción, con capa/s oculta/s con función de activación $f_{act}(x) = tanh(x)$ combinadas con neuronas lineales y salida lineal, entrenadas con 5, 10 y 20 puntos.

Podemos ver en la figura 0.7 que en el intervalo [1,0; 50,0] aproxima por extrapolación con un error menor a los demás en comparación. Por otro lado en la figura 0.8 a pesar de ser FFNN construidas específicamente con distintas cantidades de neuronas en la capa oculta, para mejorar el comportamiento en el entorno de interpolación, observamos que al extrapolar, la aproximación, se ve fuertemente afectada por la neurona lineal presente en la capa oculta, generando un error caracterizado por la función $err(x) \approx \ln(x) - x$.

Escenario 2: Aproximación de $f(x, y) = x * y$

En los inicios de la materia, vimos el "¿Por / Para que usamos redes neuronales?"; dentro de las características veíamos que son tolerantes a errores, tienen la capacidad de aprender y generalizar con poca información, etc; capacidades que también vemos en biología en los seres con sistemas nerviosos. Junto con esto, también vimos que (así como los seres vivos) no son muy buenas o eficientes para hacer cálculos exactos (esa no es su finalidad). Por lo que con este ejemplo no pretendo mostrar lo contrario. La intención no es usar FFNN para multiplicar dos números. Sin embargo este ejemplo es útil a fines ilustrativos, respecto del tema: extrapolación con FFNN y como afecta la selección de la función de activación.

¿Por que elegimos la función $f(x,y) = x * y$?

Básicamente porque vimos que se puede representar exactamente por una FFNN de una capa oculta.

Veamos que se puede escribir la $f(x,y) = x * y$ en términos de sumas de constantes y funciones un-arias (estructura general característica de cualquier FFNN).

Sabemos que la identidad $a^2 - b^2 = (a+b) * (a-b)$ con $a, b \in \mathbb{R}$.
Digamos que $x = a + b$ y que $y = a - b$.
Luego $x * y = (a+b) * (a-b)$ ie. $x * y = a^2 - b^2$.
Escribamos a en función de x, lo mismo con b.

$a = x - b$

$[b = a - y] \equiv [b = (x - b) - y] \equiv \left[b = \frac{x-y}{2}\right]$

ie. $\left[a = x - \frac{x-y}{2}\right] \equiv \left[a = \frac{x+y}{2}\right]$

por lo tanto $x * y = (\frac{x+y}{2})^2 - (\frac{x-y}{2})^2$.

De aquí que podemos calcular exactamente $f(x,y) = x * y$ con una FFNN de una capa oculta de solo 2 neuronas, con función de activación $f_{act}(x) = x^2$ y capa de salida lineal.

En teoría es posible, aunque en la practica (entrenando la FFNN con el algoritmo backpropagation) no lo conseguí; pero si lo conseguí con una red FFNN con 3 neuronas en la capa oculta.

Aproximación de $f(x,y) = x * y$

Entrenamos sobre 10 puntos de la función $f(x,y) = x * y$ tomados aleatoriamente en el intervalo $[-10;10];[-10;10]$ en $(x;y)$. Imponiendo un limite $\epsilon = 0{,}02$ de tolerancia de error de aprendizaje por patrón.

Aproximamos con las siguientes FFNN:

- FFNN típica de una capa oculta con 12 neuronas con función de activación $tanh(x)$ y capa de salida lineal.
- FFNN con una capa oculta de 3 neuronas con funciones de activación $f_{act}(x) = x^2$ y capa de salida lineal.
- FFNN obtenida de un algoritmo de construcción, con una capa oculta con función de activación $f_{act}(x) = tanh(x)$ combinadas con neuronas lineales y salida lineal. (El algoritmo dio como resultado solo una capa oculta de 110 neuronas)

Como en el caso anterior haremos los gráficos de la diferencia $g(x_j, y_j) - \hat{g}(x_j, y_j)$ (erro de aproximación). Donde $g(\cdot)$ es la función a aproximar por FFNN y $\hat{g}(\cdot)$ es la estimación de $g(\cdot)$ dada por FFNN; midiendo el error en el intervalo $[-100;100];[-100;100]$ en $(x;y)$.

A continuación vemos la Figura 0.9, las curvas de error de aproximación de interpolación y extrapolación. (aproximación dada por FFNN típica con una capa oculta de 12 neuronas de función de activación $tanh(x)$ y capa de salida lineal)

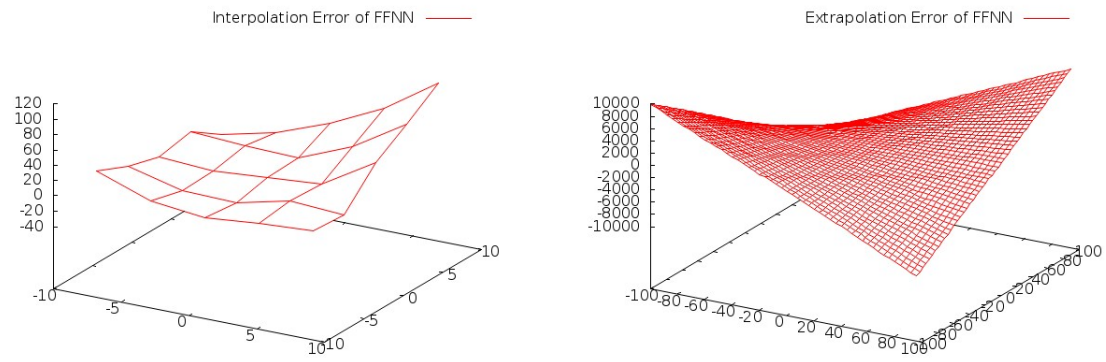

Figura 0.9: Error de aproximación a $f(x, y) = x * y$ por FFNN típica con una capa oculta de 12 neuronas de función de activación $tanh(x)$ y capa de salida lineal, entrenada con 10 puntos en el intervalo $[-10; 10];[-10; 10]$.

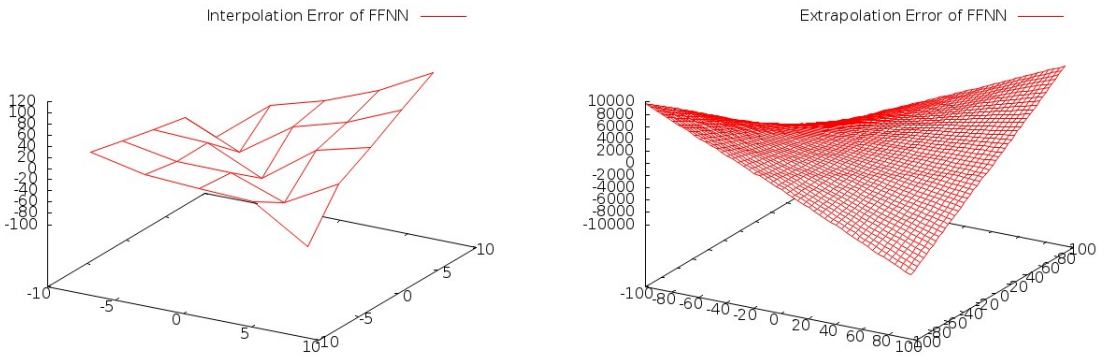

Figura 0.10: Error de aproximación a $f(x, y) = x * y$ por una FFNN obtenida de un algoritmo de construcción, con una capa oculta con función de activación $f_{act}(x) = tanh(x)$ combinadas con neuronas lineales y salida lineal. (El algoritmo dio como resultado solo una capa oculta de 110 neuronas), entrenada con 10 puntos en el intervalo $[-10; 10];[-10; 10]$.

En las figuras 0.9 y 0.10 podemos observar (en ambas) que la interpolación de por si es mala, ya que las gráficas de error en el inérvalo $[-10; 10];[-10; 10]$ alcanza valores entre -100 a 120, siendo que $f(x, y) = x * y$ en ese rango alcanza esos valores y en los puntos de entrenamiento se pidió una tolerancia limite de

$\epsilon = 0{,}02$ (la cual se respeto, pero estos valores solo se dan en entornos muy cercanos a los puntos de entrenamiento).
En ambas gráficas de extrapolación vemos que el error es total. (Error caracterizado por la función $err(x, y) \approx x * y$ que es la propia función objetivo.) Es decir, ninguna de las FFNN aproxima a la función objetivo.

A continuación veremos el error de aproximación por la red de una capa oculta de 3 neuronas con función de activación $f_{act}(x) = x^2$.

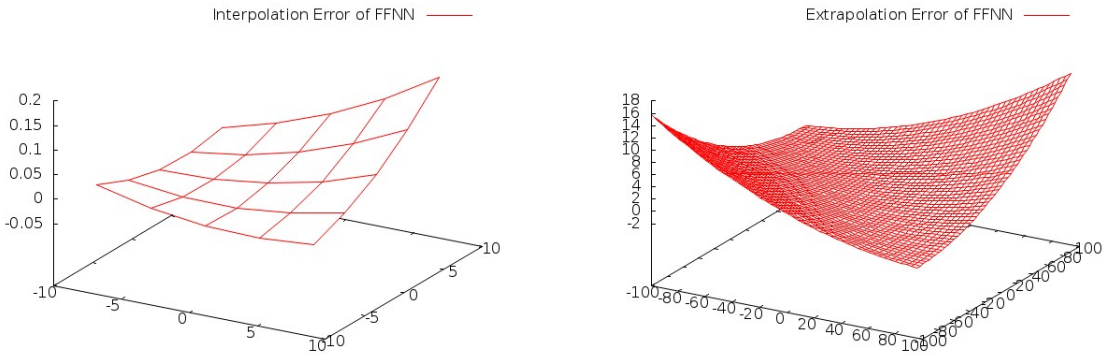

Figura 0.11: Error de aproximación a $f(x, y) = x * y$ por FFNN con una capa oculta de 3 neuronas de función de activación $f_{act}(x) = x^2$ y capa de salida lineal, entrenada con 10 puntos en el intervalo $[-10; 10];[-10; 10]$.

Podemos ver que las FFNN con capa oculta $f_{act}(x) = x^2$ interpola y extrapola muy bien toda la región. Las gráficas de error en el intervalo $[-10; 10];[-10; 10]$ alcanza valores entre -0.05 a 0.2, siendo que $f(x, y) = x * y$ en ese rango alcanza valores entre -100 y 100 (error de interpolación en $[-10; 10];[-10; 10]$ del $\%0.2$). En la gráfica de extrapolación vemos que el error de extrapolación alcanza valores entre -2 a 18, siendo que $f(x, y) = x * y$ en ese rango alcanza valores entre -10000 y 10000 (error de extrapolación en $[-100; 100];[-100; 100]$ del $\%0.18$).

Conclusiones

1. A medida que aumenta la cantidad elementos en el conjunto de entrenamiento vemos que el error se acerca más a 0.

2. En algunos casos, cuando la arquitectura de la red o la/s función/es de activación de las neuronas no son apropiadas, se puede dar el efecto de sobre entrenamiento, en donde la red aprende bien todos los valores de entrenamiento, pero no interpola ni extrapola bien.

3. La estructura de la red (en cuanto a cantidad de capas y numero de neuronas por capa) pude ser determinante para la interpolación y una estructura adecuada puede ayudar a tener un mejor entrenamiento para cierto conjunto de entrenamiento; pero desde el punto de vista de la extrapolación, notamos que la función de activación tiene un rol mucho mas significativo.

4. El algoritmo backpropagation tiene limitaciones, como por ejemplo que con FFNN entrenadas con dicho algoritmo, no podremos aproximar bien funciones discontinuas cerca de puntos de discontinuidad. (existen alternativas como los algoritmos evolutivos o geneticos aplicables a FFNN)

5. Notamos que en general las aproximaciones son buenas siempre y cuando los datos del conjunto de entrenamiento, estén bien distribuidos en el espacio objetivo que queremos representar. En este caso vemos que la interpolación es buena pero la extrapolación no siempre lo es.

6. Mientras mas características comparten, la función resultante de la composición de las funciones de activación de las distintas capas de la FFNN, con la función objetivo:

   a) Con pocos puntos de entrenamientos se puede lograr un buen nivel de interpolación.

   b) La FFNN puede ser mas pequeña en cantidad de neuronas y seguir siendo buena aproximación para la función objetivo.[1]

   c) Mejora significativamente el alcance de extrapolación, con un error menor.

[1] Mientras mas grande es la red, por lo genera mas fácil sera el entrenamiento (dentro de parámetros razonables)

Bibliografía

- Introduction to the Theory of Neural Computation (Lecture Notes Vol 1) - John Hertz, Anders Krogh, Richard G. Palmer.
- A New Strategy for adaptively Constructing Multilayer FeedForward Neural Networks - L. Ma, K. Khorasani
- Comparación del Desempeño de Funciones de Activación en Redes Feedforward para aproximar Funciones de Datos con y sin Ruido - Luis Llano, MSc.(c)1, Andrés Hoyos, Est.2, Francisco Arias, Est.2, Juan Velásquez, PhD.(c)3 1 Interconexión Eléctrica S.A. E.S.P. (ISA), Colombia 2 GIDIA: Grupo de Investigación y Desarrollo en Inteligencia Artificial 3 Grupo de Finanzas Computacionales Escuela de Ingeniería de Sistemas, Facultad de Minas Universidad Nacional de Colombia Sede Medellín
- http://bluetiger.bauchlandung.org/other/annea/
- http://en.wikipedia.org/wiki/Feedforward_neural_network