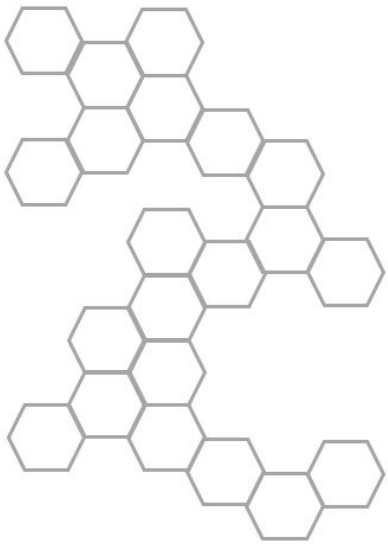
ITBA | Sistemas de Inteligencia Artificial

Redes Neuronales

[NEURAL NETWORK]



(54.623) Juan Dantur
(55.382) Ariel Debrouvier
(53.396) Agustín Golmar

❖ Tabla de Contenidos

1. Introducción

En el presente informe, se describe la implementación de una red neuronal multicapa con aprendizaje supervisado.

Además, como requerimiento del trabajo práctico se utilizó la red para resolver un problema que consiste en aproximar la función de altura de un terreno (Figura 4.1.2). Se realizaron diversas pruebas con distintas arquitecturas y variando los parámetros de las mejoras implementadas.

2. Implementación

A continuación se detallarán las características funcionales y no-funcionales del sistema desarrollado, junto con las decisiones de diseño tomadas.

Se realizó una aplicación en *Matlab*, la cual recibe desde un archivo de configuración llamado *nerve.json* los parámetros variables de la red neuronal (cantidad de capas, neuronas en cada capa, etc.). La aplicación consiste de las siguientes clases y en la *Figura 4.1.1* se detalla un esquema representativo:

- **Nerve:** Es la clase principal. Se encarga de instanciar al Perceptrón, almacenar los resultados de entrenamiento y testeo del mismo y obtener los respectivos errores cuadráticos medios.
- **Perceptron:** Esta clase contiene a la red neuronal. Posee métodos para entrenar a la red, testear, acceder a cada una de las capas, modificar los pesos, la tasa de aprendizaje y el momentum.
- **Layer:** Cada instancia de esta clase representa una capa de neuronas. Cada capa tiene métodos para activar sus neuronas y modificar sus pesos.
- **Configurator:** Esta clase se encarga de leer el archivo json con los parámetros mediante la librería *JSONLab* así como el archivo con los datos del terreno.
- **Generator**: Se encarga de generar patrones para entrenamiento y para testeo en cada iteración.
- **Logger:** Imprime en pantalla todos los parámetros de interés para el análisis de la red.
- **OutputGrapher:** Genera un gráfico del terreno en base a los puntos dados, y otro en base a los puntos aprendidos por la red.
- **Processor**: Normaliza los datos para que estén dentro de un intervalo acotado.
- **TransferFactory:** Contiene implementaciones de diversas funciones de activación, así como sus respectivas derivadas.

❖ **Validación Cruzada (Cross-validation)**

Se diseñaron 3 metodologías de validación cruzada, las cuáles permiten modificar la forma en la que se desarrolla la red neuronal, a lo largo del entrenamiento:

- **Joint**: consiste en seleccionar en cada iteración un subconjunto de entrenamiento y otro de testeo, ambos disjuntos y seleccionados al azar

del total de instancias disponibles con distribución uniforme. El proceso se repite en cada generación y por lo tanto, con un gran número de épocas, la red convergerá a una aproximación en la cual todas las instancias fueron utilizadas al menos una vez, tanto para testeo como para entrenamiento.

Debido a ello, no puede apreciarse el fenómeno de *overfitting*. Esto es causado por el hecho de que los conjuntos seleccionados son disjuntos durante una generación, pero no durante el proceso global de aprendizaje.

- **Disjoint**: para solventar la falta de apreciación del *overfitting*, se construyó esta validación, la cual selecciona un par de conjuntos disjuntos al igual que *joint*, pero retiene siempre estas versiones para generar nuevos pares de subconjuntos en las sucesivas épocas. Esto permite operar de forma análoga a *joint*, pero gracias a que las fuentes de instancias se mantienen separadas, poco a poco las curvas de error cuadrático medio se alejarán una de otra.

- **Manual**: consiste en seleccionar de forma manual un subconjunto de instancias, ya sea para regular la granularidad del espacio muestral, o para entrenar una red en una dirección o zona particular. Luego de seleccionar la muestra a utilizar, se puede operar bajo cualquiera de las validaciones anteriores (*disjoint* o *joint*). Ocasionalmente, una mala selección de instancias puede generar una nueva divergencia en la información ya aprendida por la red, *i.e.*, la red olvida patrones aprendidos.

❖ **Pre-procesamiento**

- **Bits**: En caso de que las instancias sean binarias, los datos se procesan para que valgan 1 o -1 y sean compatibles con la función de activación *sign* o *heaviside*. Este pre-procesamientos se utilizó al construir la red para verificar su correcto funcionamiento (específicamente, se aprendieron las funciones *AND*, *OR*, *XOR* y *ADD*).

- **Min-Máx:** Para cualquier rango de valores, se normalizan los datos teniendo en cuenta el máximo y el mínimo y ubicándolos en el intervalo [-1,1] para aprovechar la función *tanh*. Es el pre-procesamiento ideal seleccionado para el terreno solicitado.

- **Half Min-Máx:** Para cualquier rango de valores, se normalizan los datos al intervalo [0,1] con el objetivo de usar la función *exp*. Sin embargo, la arquitectura propuesta no pudo realizar un aprendizaje adecuado mediante esta función de transferencia.

❖ **Optimizaciones**

Se implementaron las siguientes optimizaciones:

- **Vanishing Gradient Problem**: este problema ocasionado por la técnica del gradiente descendente durante el proceso de *backpropagation*, satura las capas de entrada, impidiendo que estas aprendan información adicional, lo que ocasiona un estancamiento en las capas subsiguientes.

Luego de analizar la forma en la que operan las diferentes magnitudes de la constante agregada a las variables δ para mitigar el problema, se hizo notoria la necesidad de implementar un algoritmo *anti-vanishing* que permita actualizar de forma dinámica y automática el valor de la constante. Esto se debe a que si se retiene el mismo valor durante todo el proceso de aprendizaje, la red alcanzará un límite máximo de variación, y el aprendizaje se estancará nuevamente.

Otra medida para mitigar el problema es utilizar una arquitectura de poca profundidad (pocos *layers*), o cambiar la función de activación empleada por una que no presente las mismas características (como la función *ReLU*).

- **Learning Momentum**: se destacó durante el análisis que en el caso del momento inercial de aprendizaje, inclusive pequeños valores producen un efecto satisfactorio en la velocidad de aprendizaje, y en la caída del error. Sin embargo, y a diferencia del resto de parámetros, la selección de un valor demasiado elevado satura de forma significativa la red completa, e impide una recomposición de la capacidad de aprendizaje de la misma.

  En definitiva, la saturación grave a través del momento de aprendizaje puede dejar obsoleta una red neuronal completa.

- **Adaptive Learning Rate**: se observó una reducción en el aprendizaje (a través de la integral de la curva, es decir, el área que encierra), a medida que la ventana de tendencia se aumentaba. Adicionalmente, se observa que la variación del aprendizaje posee un punto de inflexión en la cual la red cambia su estado inicial: o intenta aprender lo más que pueda las primeras generaciones, o se muestra reticente al cambio (esto último se hace más notorio conforme la ventana se incrementa lo suficiente).

- **Pattern/Weight Noise Injection**: se determinó que la introducción de ruido tanto en patrones como en matrices de pesos, puede llegar a interferir con el proceso de aprendizaje.

❖ Arquitectura Óptima

Para determinar la arquitectura óptima, se realizaron pruebas variando la cantidad de capas ocultas, la cantidad de neuronas en cada una y los parámetros de las mejoras, entre otras variables. En todos los casos se tuvo en cuenta los problemas que ocasiona el desnivel del orden de magnitud en el aprendizaje o la estabilidad de la red.

Debido a que el espacio de búsqueda (en referencia al espacio de los valores permitidos en cada uno de los parámetros), es altamente extenso, la optimalidad se considera de forma subjetiva y no absoluta. Dentro de las arquitecturas seleccionadas, y de las estimaciones realizadas, el modelo seleccionado fue el de mayor *performance*.

3. Resultados

Debido a la gran cantidad de parámetros y arquitecturas posibles, debieron realizarse múltiples pruebas para determinar una configuración con la cual la red aprenda de la mejor manera los patrones del terreno.

- Arquitectura Óptima

Las pruebas sobre las distintas arquitecturas, que se realizaron incrementando la cantidad de neuronas por capa y luego la cantidad de capas, determinaron que la configuración que obtiene mejores resultados es con dos capas ocultas, con 100 y 45 neuronas en cada una. (Tabla 4.2.1.)

A partir de esta arquitectura, se decidió comenzar a probar los parámetros de las optimizaciones que sean óptimos y reduzcan el error cuadrático medio aún más.

- Optimizaciones

Inicialmente se realizaron pruebas aisladas de las optimizaciones, dejando la función de activación, beta y el learning rate fijos, y probando cada optimización por separado.

En el caso del *momentum*, valores altos de alpha generaron una gran oscilación del error cuadrático medio (Figura 4.1.3.). Para definir el valor de alfa que permite aprender mejor a la red, se probó sobre la arquitectura deseada, con distintos valores y se obtuvo menor error cuadrático medio con un alfa de 0.1 (Figura 4.1.2).

La mejora que soluciona el *Vanishing Gradient Problem***,** también requirió de pruebas que dieron como resultado que un vanishing limit de 3x10^-6, evita que no haya variaciones cuando la actualización de pesos tiende a ser nula. (Tabla 4.2.3.).

Agregando *pattern noise* de alta interferencia, noise level de 0.5 y chance del 50%, se producen grandes desplazamientos sobre la red con respecto al aprendizaje deseado y el error oscila de una manera pronunciada.
Si se agrega una baja interferencia de 0.1 y una chance del 25%, se exhibe un comportamiento más normal, pero en el largo plazo, el error se estabiliza y no se consigue una notable disminución del mismo. (Tabla 4.2.4.)

El *weight noise,* concluímos que puede llegar a provocar saturación en las neuronas en caso de que el mismo sea de 0.1 y una chance del 25%. Se optó por agregar ruido del orden de 1x10^-2 y una chance de 2.5% para no interferir con el proceso de aprendizaje.

Para el *learning rate adaptativo,* el parámetro a definir fue la ventana o cantidad de pasos K, para la cual se considera que el error crece consistentemente. Las "caídas" del error, provocan que la tasa aumente. A medida que se aumentó la cantidad de pasos, la tasa de aprendizaje tendió a anularse y mientras más pasos se consideren, menor era la chance de que vuelva a crecer.

Las pruebas determinaron que un k = 2 (Figura 4.1.5), con a = 0.035 y b = 0.045, se logra adaptar el learning rate, permitiendo que cuando el error disminuye, se aprenda a mayor velocidad.

Tras analizar las múltiples mejoras, y seleccionar las que optimizaron el proceso de aproximación se realizó la predicción que se ve en la Figura 4.1.6.

- ❖ Funciones de activación

Por otro lado, se limitó la cantidad de épocas y se ejecutó la red, dejando constante todos los parámetros menos la función de activación, para de esta manera comparar las funciones sigmoideas tangencial hiperbólica y exponencial. La función tangencial consiguió mejores resultados para distintas arquitecturas, consiguiendo un error cuadrática medio menor. La función exponencial, requiere de una cantidad de épocas mucho mayor que la otra opción, por lo que se descartó como función de activación para la arquitectura óptima (Tablas 4.1.4, 4.1.5, 4.1.6, 4.1.7).

- ❖ Análisis

Alguno de los principales aspectos a resaltar analizando los resultados de las pruebas:

- Agregar más capas a la red no redujo el error, sino que fue más efectivo tener menos capas pero con una gran cantidad de neuronas (Tabla 4.2.1).

- Debido a la complejidad del terreno, los puntos correspondientes a los máximos de la función, son los que a la red le cuesta más aprender, principalmente debido a que son pocos los patrones correspondientes a estas regiones del terreno.

- El error baja considerablemente en las primeras épocas, pero una vez que se cruza el umbral de las 500 épocas, el mismo tiende a estabilizarse.

- Agregar ruido tanto a los patrones como a los pesos afecta negativamente al aprendizaje de la red neuronal, provocando desplazamientos de toda la función y un aprendizaje incorrecto.

4. Anexo

En esta sección se despliegan las figuras referenciadas y los resultados numéricos obtenidos durante la etapa de *testing* y *benchmarking*.

4.1. Figuras

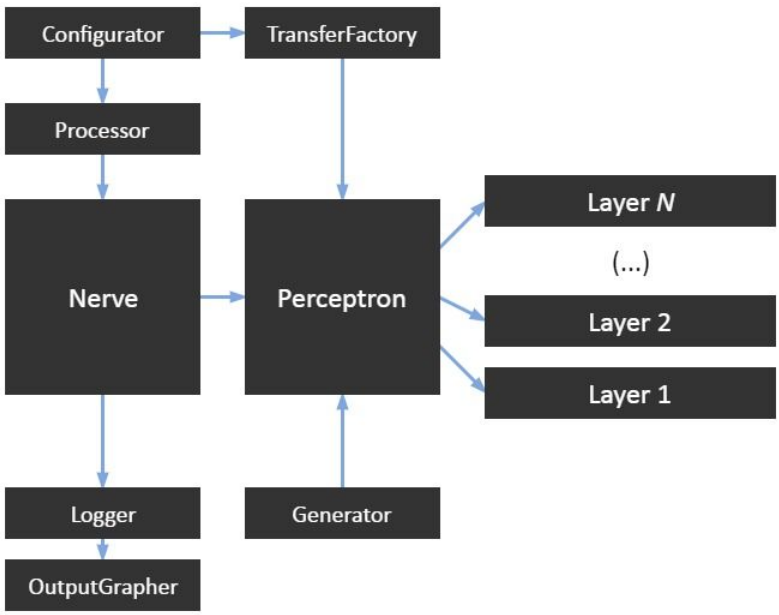


Figura 4.1.1: Representación de las diversas clases que se implementaron

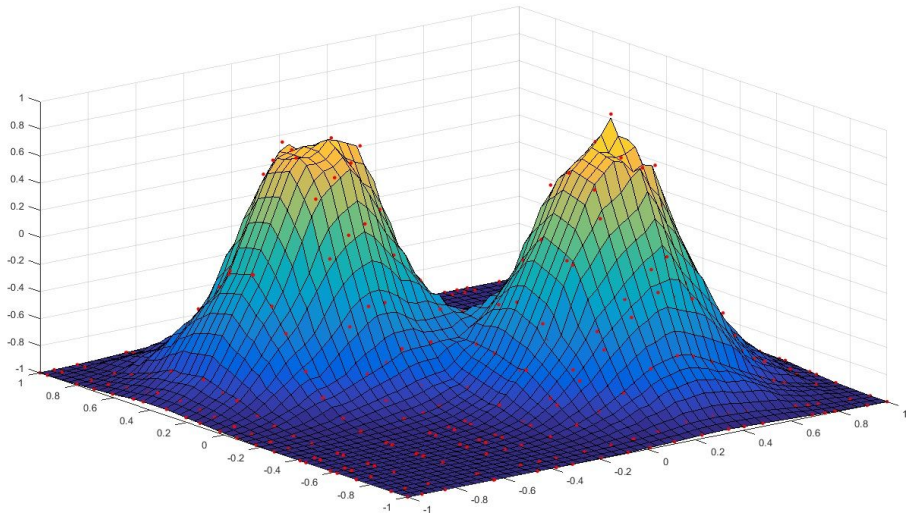


Figura 4.1.2 :Interpolación del terreno .

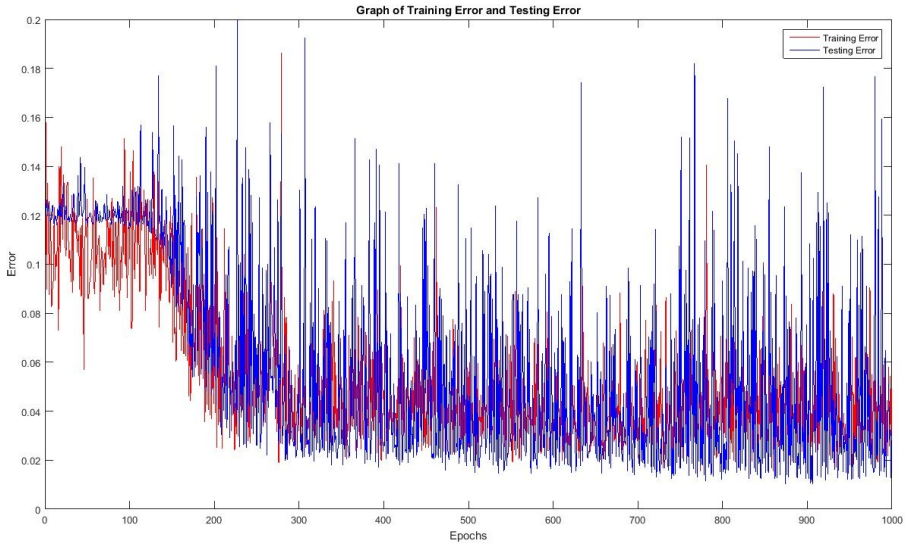


Figura 4.1.3: Variación de los errores de entrenamiento y testeo con constante de momentum = 0.8.

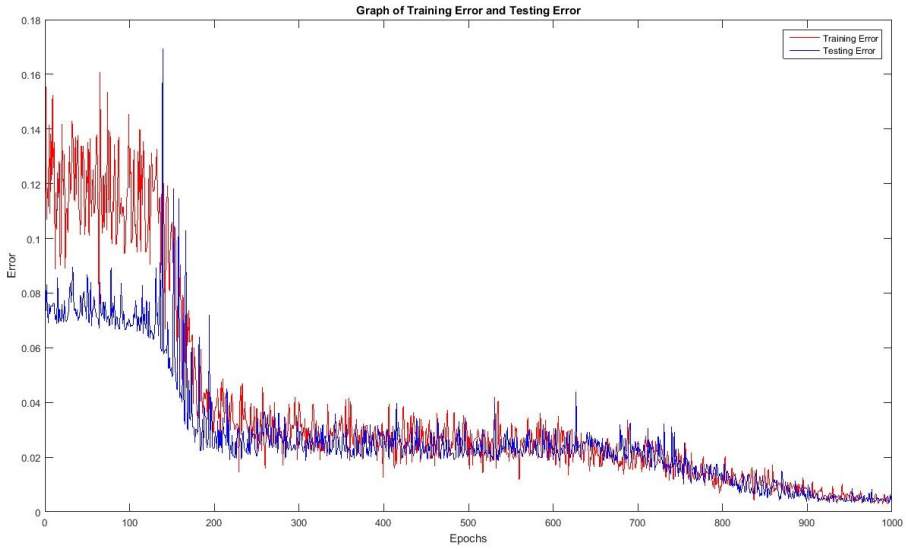


Figura 4.1.4: Variación de los errores de entrenamiento y testeo con constante de momentum = 0.1.

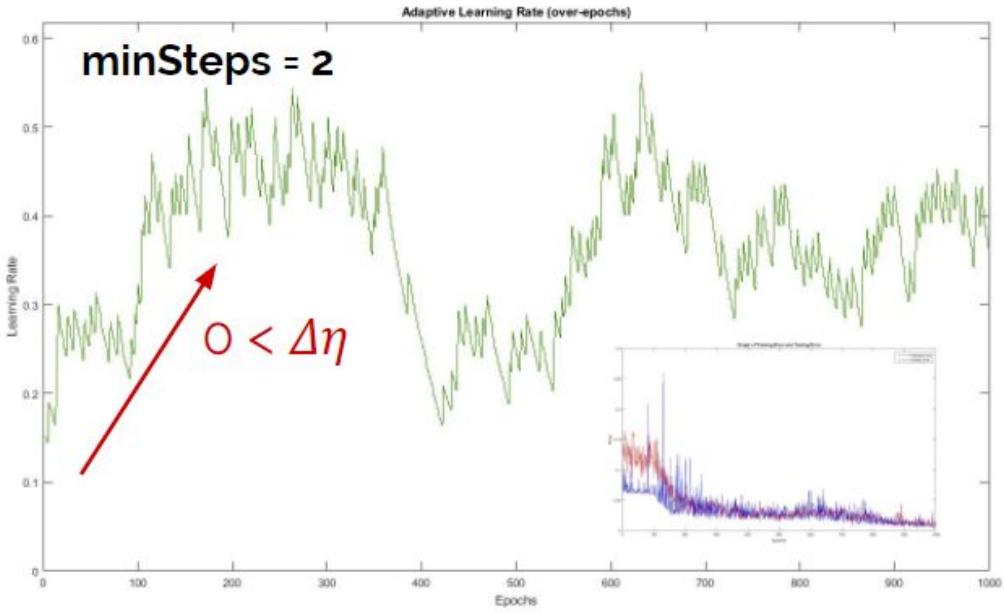


Figura 4.1.5 : Fluctuación de learning rate con k = 2.

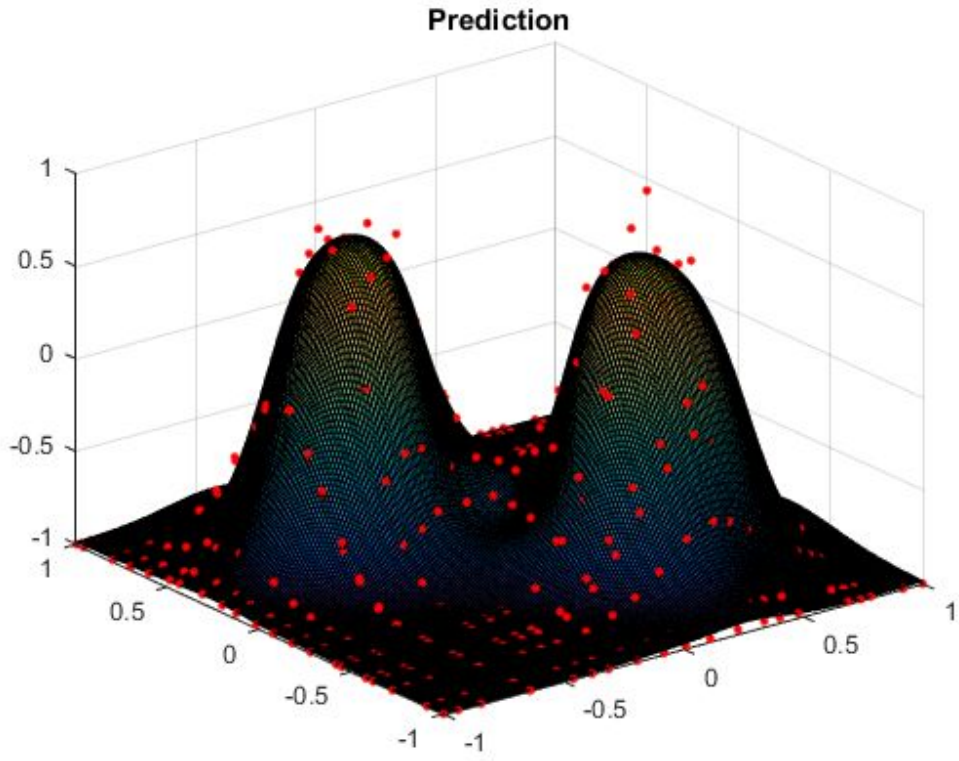


Figura 4.1.6. Terreno original (puntos rojos) vs predicción de la arquitectura óptima (superficie)

4.2. Benchmarks

Arquitectura	**ECM**	**Épocas**	**Tiempo**
[2 15 1]	0.038	1000	131.35
[2 15 10 1]	0.025	1000	167.88
[2 40 20 1]	0.003	1000	153.02
[2 100 45 1]	0.019	1000	182.63
[2 20 15 10 1]	0.045	1000	197.55

Tabla 4.2.1: Comparación de arquitecturas

Arquitectura: [2 100 45 1] **Beta:** 0.65 **Tasa de Aprendizaje:** 0.15 **Activación:** tanh			
Error cuadrático medio	Épocas	Tiempo	Momentum
0.03	1000	150.24	Alfa: 0.01
0.002	1000	186.85	Alfa: 0.1
0.012	1000	154.34	Alfa: 0.4
0.02	1000	161.5	Alfa: 0.8

Tabla 4.2.2: Comparación de valores de alfa sobre arquitectura elegida

Arquitectura: [2 100 45 1] **Beta:** 0.65 **Tasa de Aprendizaje:** 0.15 **Activación:** tanh			
Error cuadrático medio	Épocas	Tiempo	Vanishing Limit
0.019	1000	179.87	3.5x10^-2
0.008	1000	183.57	3.5x10^-3
0.02	1000	189.64	3.5x10^-4
0.004	1000	189.33	3.5x10^-6

Tabla 4.2.3: Comparación de valores de vanishing limit.

Arquitectura: [2 100 45 1] **Beta:** 0.65 **Tasa de Aprendizaje:** 0.15 **Activación:** tanh			
Error cuadrático medio	Épocas	Tiempo	Pattern Noise
0.07	1000	196.45	Noise: 0.5 Probability: 50%
0.005	1000	182.47	Noise: 0.1 Probability: 25%
0.013	1000	199.563	Noise: 0.01 Probability: 25%

Tabla 4.2.4: Comparación de valores de ruido sobre patrones

Arquitectura: [2 100 45 1] **Beta:** 0.65 **Tasa de Aprendizaje:** 0.15 **Activación:** tanh			
Error cuadrático medio	Épocas	Tiempo	Weight Noise
0.21	1000	181.79	Noise: 0.1 Probability: 25%
0.03	1000	178.22	Noise 0.01 Probability: 10%
0.006	1000	180.25	Noise: 0.0001 Probability: 2.5%

Tabla 4.2.5: Comparación de valores de ruido sobre pesos.

Arquitectura: [2 100 45 1] **Beta:** 0.65 **Tasa de Aprendizaje:** 0.15 **Activación:** tanh			
Error cuadrático medio	Épocas	Tiempo	Adaptative LR
0.025	1000	180.17	a: 0.035 b: 0.045 K = 1
0.004	1000	179.33	a: 0.035 b: 0.045 K = 2
0.051	1000	181.29	a: 0.035 b: 0.045 K = 5

Tabla 4.2.6: Comparación de parámetros de LR adaptativo.

Arquitectura: [2 100 45 1] **Beta:** 0.65 **Tasa de Aprendizaje:** 0.15 **Activación:** tanh			
Error cuadrático medio	Épocas	Tiempo	Adaptative LR
0.006	1000	189.29	a: 0.1 b: 0.045 K = 1
0.003	1000	183.81	a: 0.1 b: 0.045 K = 2
0.004	1000	182.93	a: 0.1 b: 0.045 K = 5

Tabla 4.2.7: Comparación de parámetros de LR adaptativo.

Funciones de activación 1500 Épocas		
[2 15 10 1]	tanh	exp
Tiempo	362.87s	345.47
ECM	0.0067	0.022

Tabla 4.2.8:

Funciones de activación 1500 Épocas		
[2 45 20 1]	tanh	exp
Tiempo	315.87	334.77
ECM	0.0027	0.033

Tabla 4.2.9:

Funciones de activación		
[2 20 15 10 1]	tanh	exp
Tiempo	399.96	380.34
ECM	0.0014	0.021

Tabla 4.2.10:

Funciones de activación		
[2 40 25 15 1]	tanh	exp
Tiempo	417.45 s	410.23
ECM	0.0017	0.033

Tabla 4.2.11: