Trabajo Práctico 2

Redes Neuronales

Autores

Nagelberg, Martin (56698) mnagelberg@itba.edu.ar

Grethe, Juan (57370) jgrethe@itba.edu.ar

Grabina, Martin (57360) mgrabina@itba.edu.ar

72.27 - Sistemas de Inteligencia Artificial

Prof. María Cristina Parpaglione

9 de octubre de 2019

Contenido

Introducción

El objetivo de este trabajo práctico fue desarrollar una red neuronal capaz de aprender una función multivariable con métodos de entrenamiento batch e incremental para luego poder simular la superficie de un terreno.

Para poder realizar esto, se procedió inicialmente a implementar un perceptrón simple (probado con patrones AND), el cual luego fue extendido a uno multicapa (probado con patrones XOR). Finalmente se desarrolló la red neuronal entera parametrizable cuyo comportamiento se describe en este documento.

La idea además es encontrar aquellos parámetros que nos permitan obtener la red óptima para este problema, es decir; aquella que cuente con menor error cuadrático medio en menor cantidad de épocas.

Problema

Una empresa de videojuegos precisa un desarrollo que pueda simular en su plataforma terrenos de diferentes partes del mundo a partir de mediciones de altura, latitud y longitud que el equipo de simulación ha tomado de los terrenos reales.
El objetivo de la red multicapa es aproximar de la mejor forma posible a partir de los datos de entrada y es por esto que fue entrenado de forma supervisada.

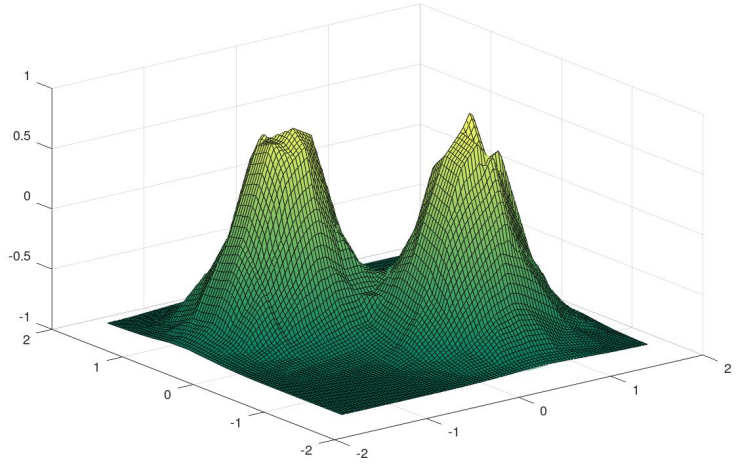


Figura 1 - Terreno real analizado

Implementación

Marco teórico

El proceso de entrenamiento busca minimizar la diferencia entre el valor obtenido y el valor esperado por cada patrón. Esto se hizo utilizando el proceso de minimización de **gradiente descendente**, que consta de dos partes, **forward propagation** y **backpropagation.** Se realiza la propagación hacia adelante y se obtiene un cierto valor de salida. Luego, como sabemos cual es el valor esperado, se corrigen los pesos. Por último, cabe destacar que para el diseño de la red neuronal, se utilizó el modelo de neurona de McCulloch-Pitts.

El código se modularizó en archivos separados para mejor mantenimiento y reutilización del mismo.
La estructura del proyecto es:

- Funciones de activación
    - Función, derivada e inversa de tan
    - Función, derivada e inversa de exp
- main.m
  Archivo central que entrena la red a partir de datos y luego genera un terreno nuevo con la información recopilada.
- configuration.m
  Archivo de configuraciones para parametrizar la red.
- data_init.m
  Archivo de inicialización de variables (v, w, d, etc.)
- incremental_trainer.m
  Implementación del aprendizaje incremental.
- batch_trainer.m
  Implementación del aprendizaje en batch
- predict.m
  Implementación de la generación de salida a partir de los datos aprendidos.
- load_terrain, plot_terrain, get_random_patterns
  Librerías auxiliares para el manejo de los datos.

Métricas clave

- Capacidad de aprendizaje
- Error de aprendizaje
- Épocas que tarda en llegar a un error razonable (E = 0.002)
- Tiempo de ejecución (no es esencial)

Casos estudiados y resultados

Se realizaron distintas corridas al programa variando de a un único parámetro a la vez, dejando fija la semilla del random que inicializa los vectores del aprendizaje.
Se fijaron los parámetros:

- Batch
- Error de corte: 0.0005
- Función Tangencial
- Beta = 1.0
- ETA = 0.0005
- Sin puntos específicos añadidos para precisión
- Sin optimizaciones (ETA adaptativo, Momentum)
- Arquitectura (50 50 50 50)

Dado la gran cantidad de gráficos obtenidos, se decidió colocarlos ordenados en el apéndice del documento. A continuación se describen los resultados de las comparaciones hechas.

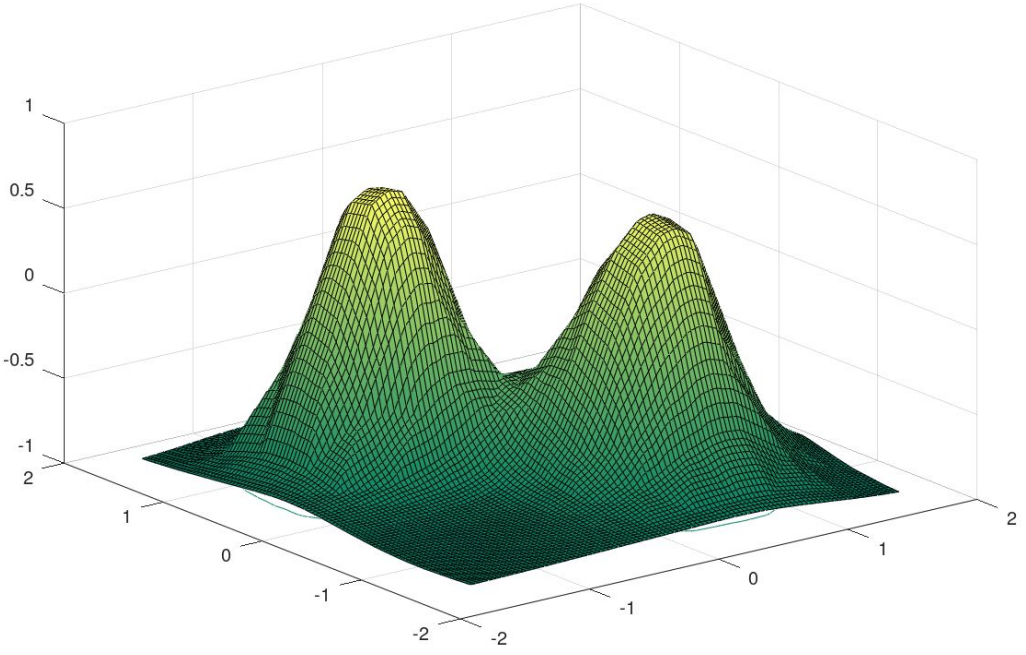


Figura 2: Terreno generado luego de 7000 épocas de aprendizaje

Batch vs Incremental

Como era de esperar, los terrenos generados por batch e incremental con parámetros iguales dieron muy parecidos, en cuanto a error y a terrenos generados, salvando que incremental demora más en su ejecución (naturalmente debido a su algoritmia).
Este comportamiento lo validamos con los siguientes papers académicos:

- https://www.researchgate.net/publication/262287657_Batch-Incremental_versus_Instance-Incremental_Learning_in_Dynamic_and_Evolving_Data
- http://www.cs.utexas.edu/~ml/papers/either-aaaisymp-92.pdf

Donde concluye que el incremental es el método más eficaz por teoría e ideal para aprendizaje en real-time (por que va acomodando los pesos todo el tiempo y se le pueden agregar datos nuevos), pero que los resultados de batch se asemejan mucho y es mucho más performante para grandes volúmenes de datos.

ETA Adaptativo

Otra de las características de la red que probamos fue la eficacia de la optimización del ETA Adaptativo, esto es, que para cada iteración, el ETA crezca o se reduzca dependiendo de si la red está aprendiendo o no (mejora o empeora su error).
En este caso utilizamos algoritmo batch, arquitectura [50 50], ETA inicial de 0.001, constante A = 0.0001, constante B = 0.05, ETA Mínimo = 0.0001 y ETA máximo = 0.005.
Los resultados fueron notables, en el caso de **con** ETA Adaptativo, la red aprendió más rápido y esto se ve reflejado en el gráfico del ECM, donde para ETA Adaptativo llama la atención una curva con mayor pendiente llegando a niveles de error bajos más rápido tal como se ve en la Figura 3, donde la curva inferior corresponde a la ejecución con la optimización activada.

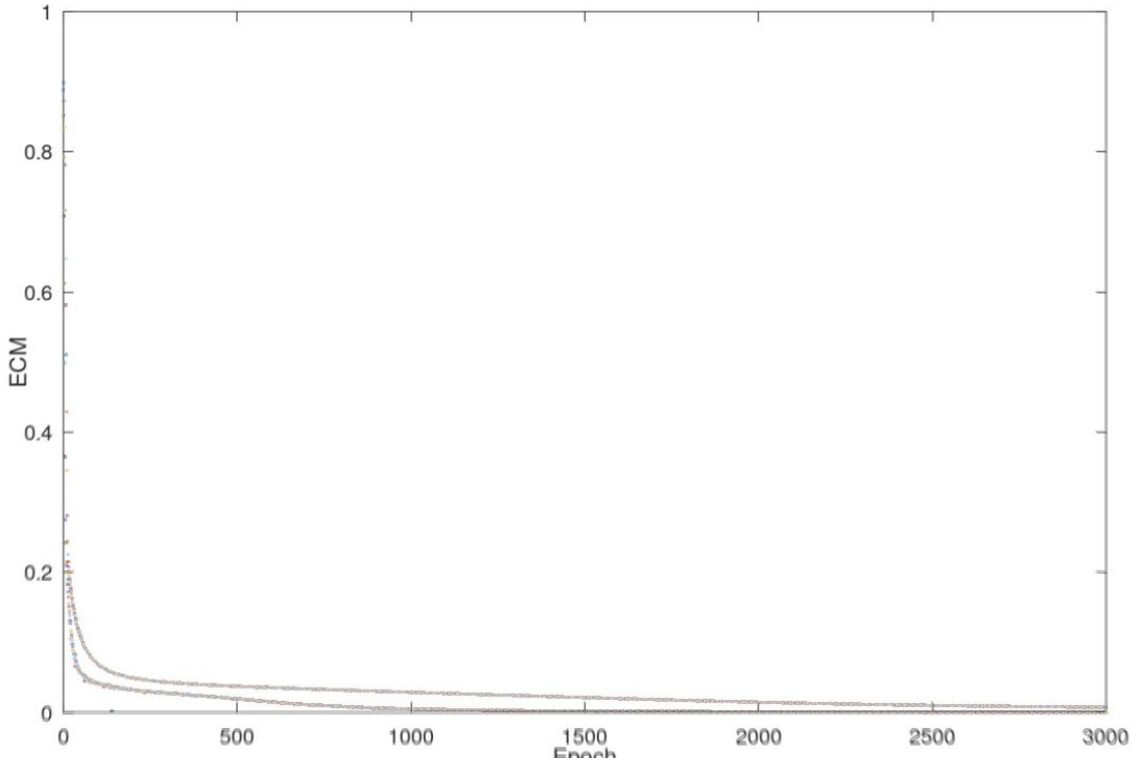


Figura 3: Transposición del ECM con y sin ETA Adaptativo
(Curva de error más baja pertenece a la ejecución con optimización).

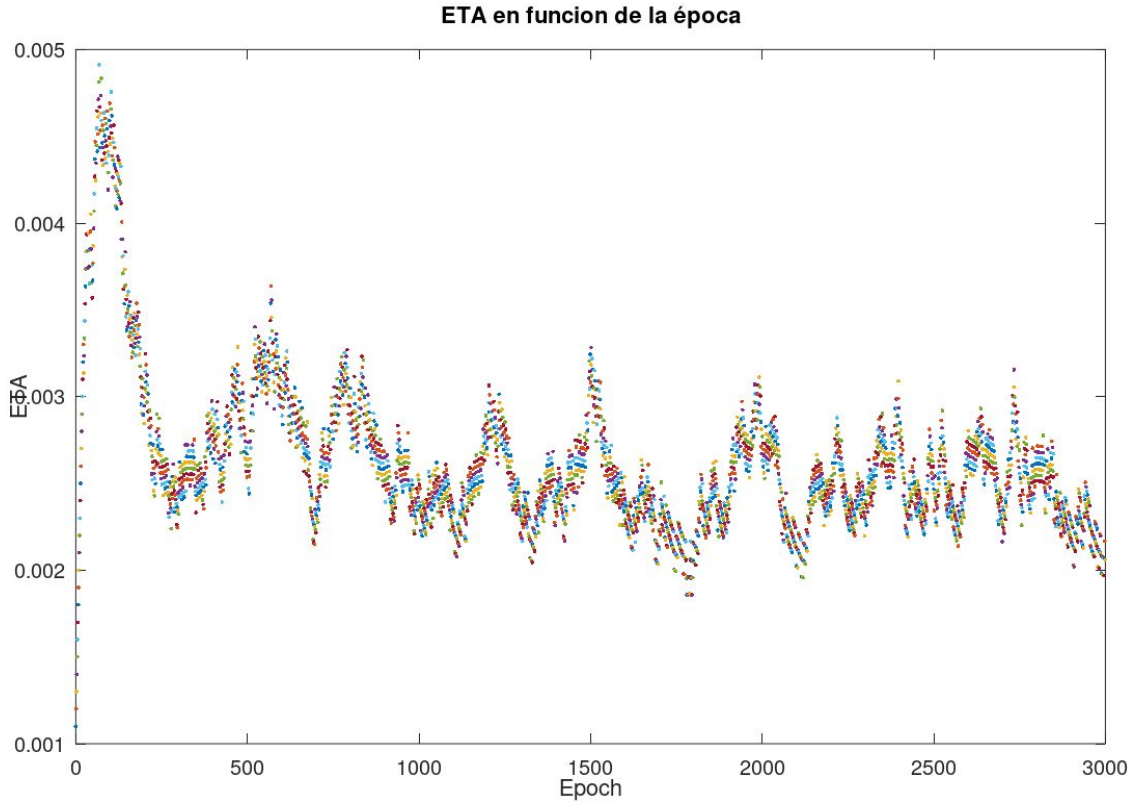


Figura 4: Evolución del ETA para la ejecución con optimización.

Momentum

Como otra metodo de optimizacion, probamos momentum. El mismo agrega un término que pesa el descenso promedio en el cálculo de los deltas, permitiendo una convergencia más rápida. En este caso utilizamos algoritmo batch, arquitectura [50 50 50 50], ETA inicial de 0.001. Los resultados fueron notables, en el caso de **no** utilizar momentum, el error cuadrático medio tuvo ciertas oscilaciones hasta estabilizarse y llegar al valor deseado. Sin embargo, al realizar esta mejora con un ALFA = 0.5, pudimos ver que las oscilaciones disminuyeron y además, que la convergencia se hace más rápidamente.

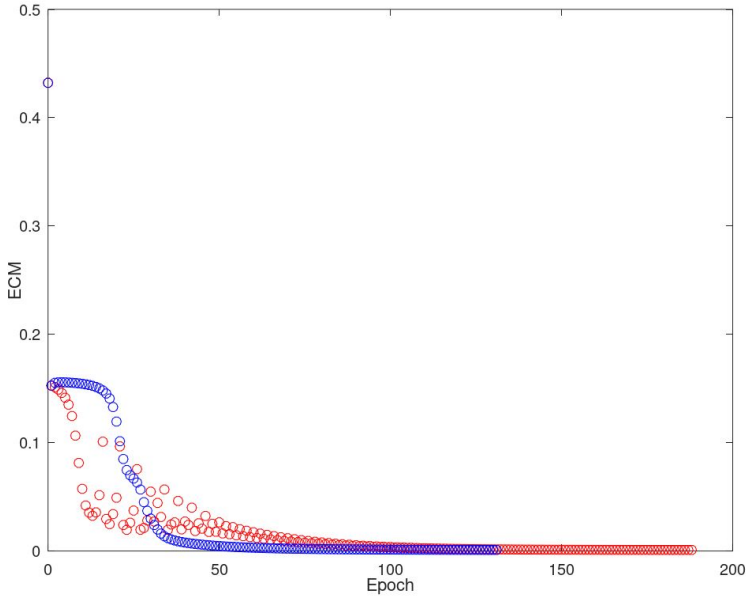


Figura 5: ECM con y sin momentum (azul y rojo, respectivamente).

Porcentaje de aprendizaje

Se comparó la capacidad de generalización de la red utilizando los mismos patrones de entrada, pero distinta cantidad. Graficamos entonces las superficies, utilizando 60% de los patrones disponibles para aprendizaje, 40%, 20%, 10% y 5%. Como se puede ver en las figuras, a partir de 20% la generalización es bastante buena, y comparando con la de 60%, no hay una gran diferencia. Sin embargo, utilizando tan solo el 10% el terreno tiene mucho error, sobretodo en los picos, pero sigue siendo aceptable. Por último, con 5% el error ya es demasiado grande y el terreno no tiene similitud con el real.

Porcentaje de aprendizaje	ECM generalización
5%	1,3813
10%	0,17512
20%	0,14385
40%	0,072681
60%	0,063448

Tangencial vs Exponencial

La Tangencial tuvo mejores resultados en la misma cantidad de épocas. La tangencial lograba llegar al error esperado más rápido, mientras que la exponencial tardaba mucho más tiempo. Si a la exponencial se la dejaba correr más tiempo, terminaban logrando resultados similares o incluso mejores.
En este punto es importante aclarar la existencia del parámetro beta, cuyo valor “óptimo” difiere según la función que se use. Como por ejemplo, beta = 1.5 funciona mucho mejor que beta = 1 para la función tangencial mientras que viceversa para la exponencial.

Beta

La modificación de la constante beta también influyó en la velocidad con la que el error converge, por ejemplo con un beta = 1.5 el sistema llegaba al error deseado en menos de 200 epochs y con un beta = 0.5 para las 1500 epochs todavía no lo había alcanzado.

Arquitectura

Cantidad de capas ocultas

Se comparó el comportamiento de la red utilizando distintas arquitecturas con similar cantidad de neuronas por capa, recordando que siempre hay dos neuronas de entrada y una de salida.
Utilizamos [50], [50 50 50] y [50 50 50 50 50], y obtuvimos resultados muy parecidos para la segunda y tercera arquitectura como se puede ver en los gráficos en el apéndice.
Nótese que en [50] terminamos la ejecución por cantidad de épocas (3000) sin llegar al error deseado, mientras que en [50 50 50] tardó aproximadamente 450 épocas y en [50 50 50 50 50] tardó aproximadamente 170 épocas (lo cual también requirió menor tiempo de ejecución), por lo tanto, a mayor cantidad de capas, el error de aprendizaje converge más rápido, pero esa disminución no es lineal y llega a un punto donde “ya no vale la pena” seguir agregando niveles. Correspondientemente con estos errores se graficaron sus terrenos generados donde, como era de esperar, la correspondiente a [50] es muy distinta de lo esperado y las otras dos son bastante aceptables.

Una capa oculta

Analizamos el comportamiento de la red utilizando una única capa oculta, con 60, 120 y 200 neuronas en esta. Sucedió que a mayor cantidad de neuronas (200) el error bajo con mayor pendiente y por lo tanto llegó a errores de aprendizaje menores en menor cantidad de épocas, teniendo en cuenta que los tres comportamientos fueron similares pero a diferentes “velocidades”. Por lo tanto, es posible solucionar el problema con una sola capa oculta, pero requiere de muchas más épocas de aprendizaje que con otras arquitecturas, y dentro del caso de una única capa oculta, a mayor cantidad de neuronas, más rápido converge el error de aprendizaje.

Red Óptima

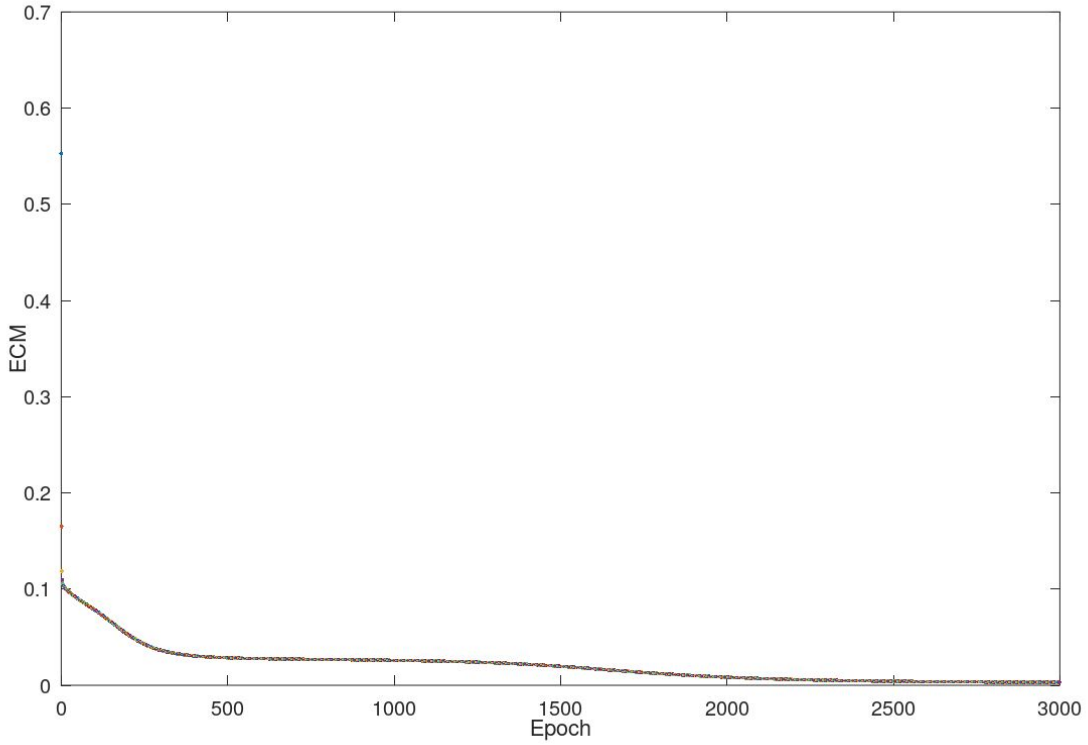
Luego de varias pruebas, elegimos como arquitectura: 2-[50 50 50]-1 dado su alto grado de precisión y rapidez de convergencia del error. Es fundamental siempre contar con las optimizaciones propuestas, incluyendo la adición de áreas de puntos especiales (como picos de montañas).

Conclusiones

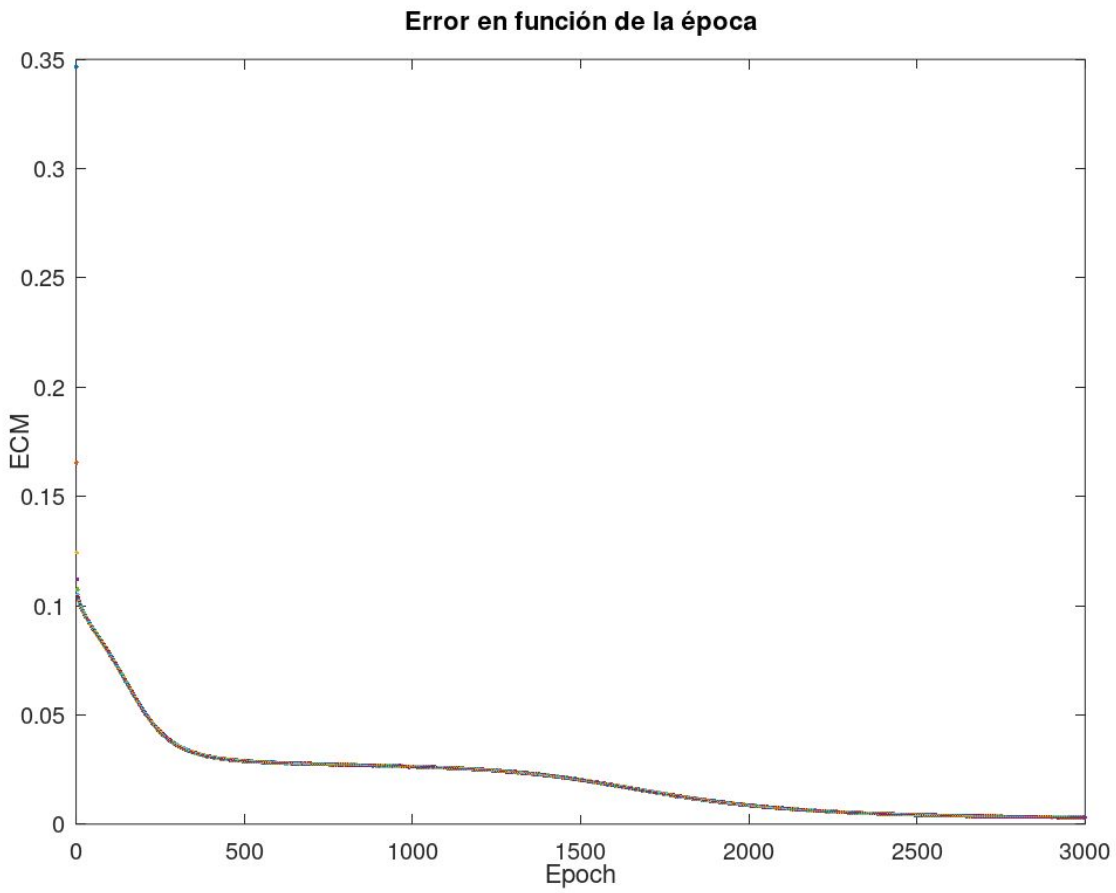
- El comportamiento de la red varía mucho en función a todos los parámetros mencionados, y los valores óptimos de los mismos también dependen de los demás para acelerar la convergencia del error.
- Siguiendo el teorema de aproximación universal (Hornik, Stinchcombe y White), pudimos ver que es posible tener una buena generalización con una única capa oculta de varias neuronas. La cantidad de capas y neuronas por capa fueron determinadas mediante la realización de pruebas empíricas
- Es importante una buena elección de patrones de entrenamiento en zonas donde la función presenta picos o valles porque es la zona donde más error puede ocurrir.
- En los segmentos lineales, la red aprende más fácilmente que en los picos/valles.
- Las optimizaciones propuestas, lograron mejorar los tiempos de convergencia.
- Respecto a la arquitectura en general, mientras más capas y neuronas utilizamos, obtuvimos mejores resultados.

Apéndice: Gráficos

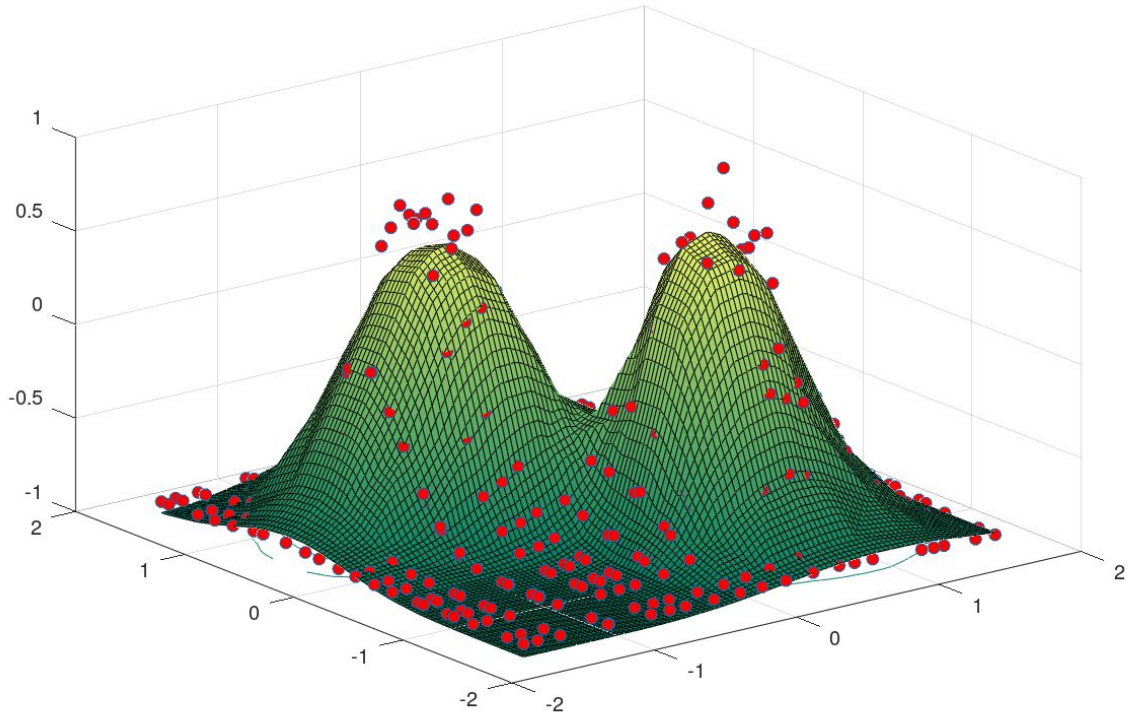
Batch vs Incremental



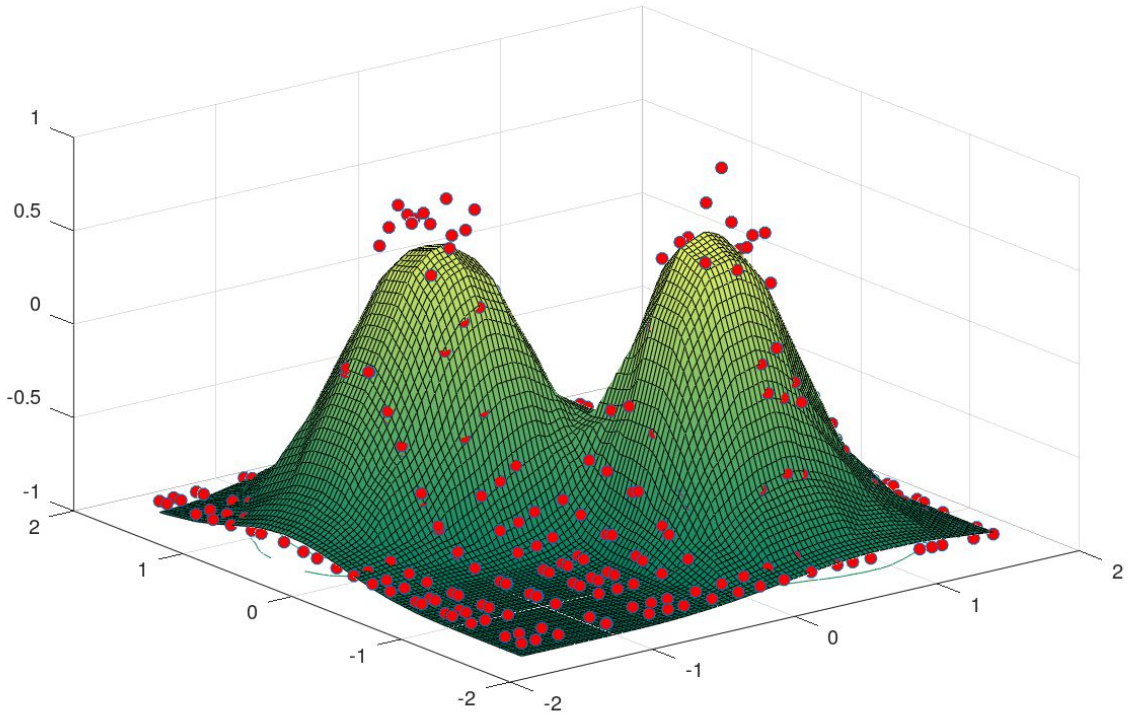
Batch



Incremental

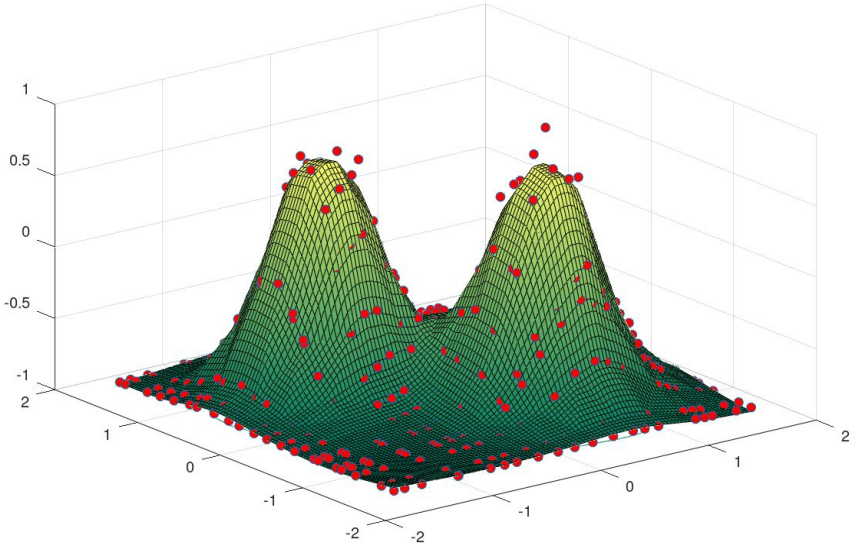


Batch

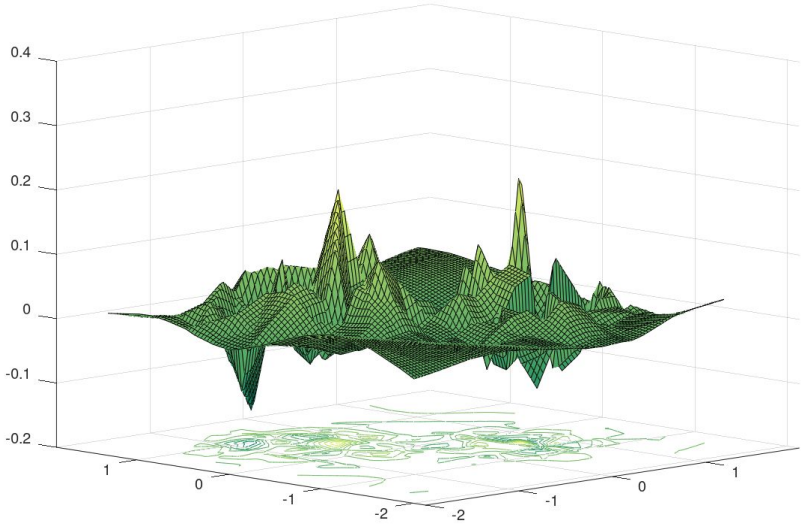


Incremental

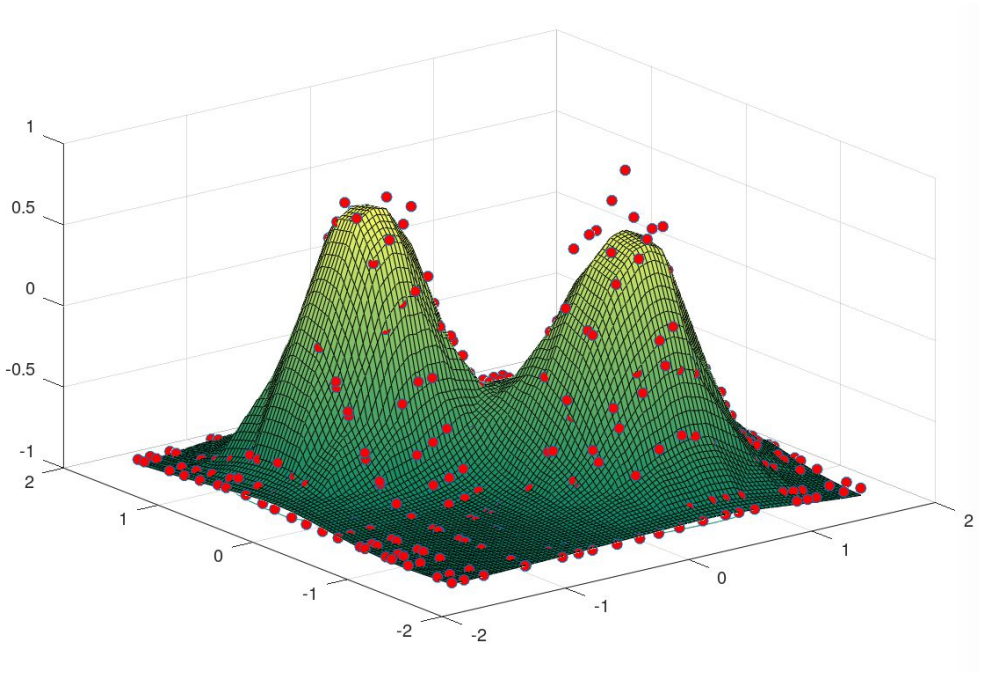
Diferencia en % de aprendizaje



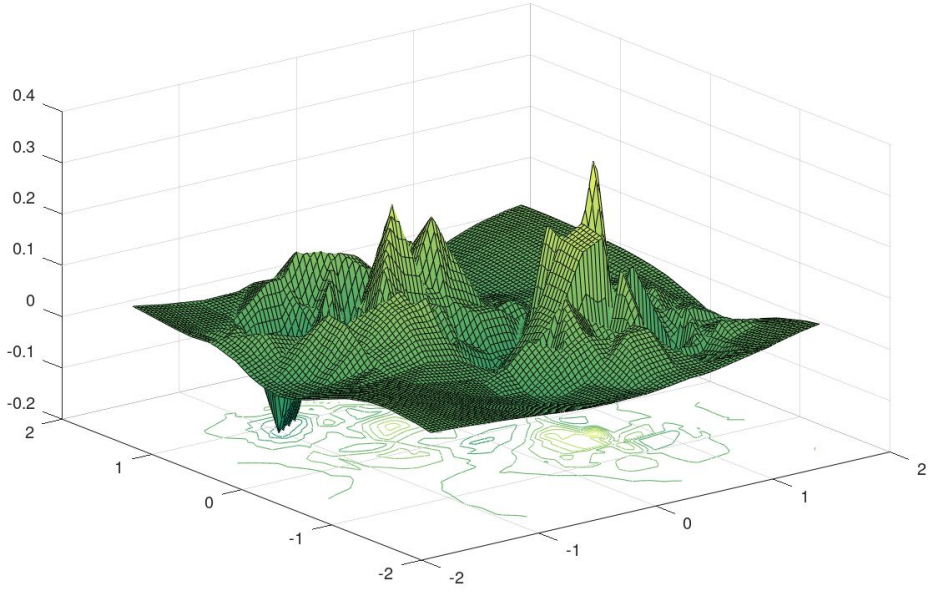
[Generalización con 60% de aprendizaje]



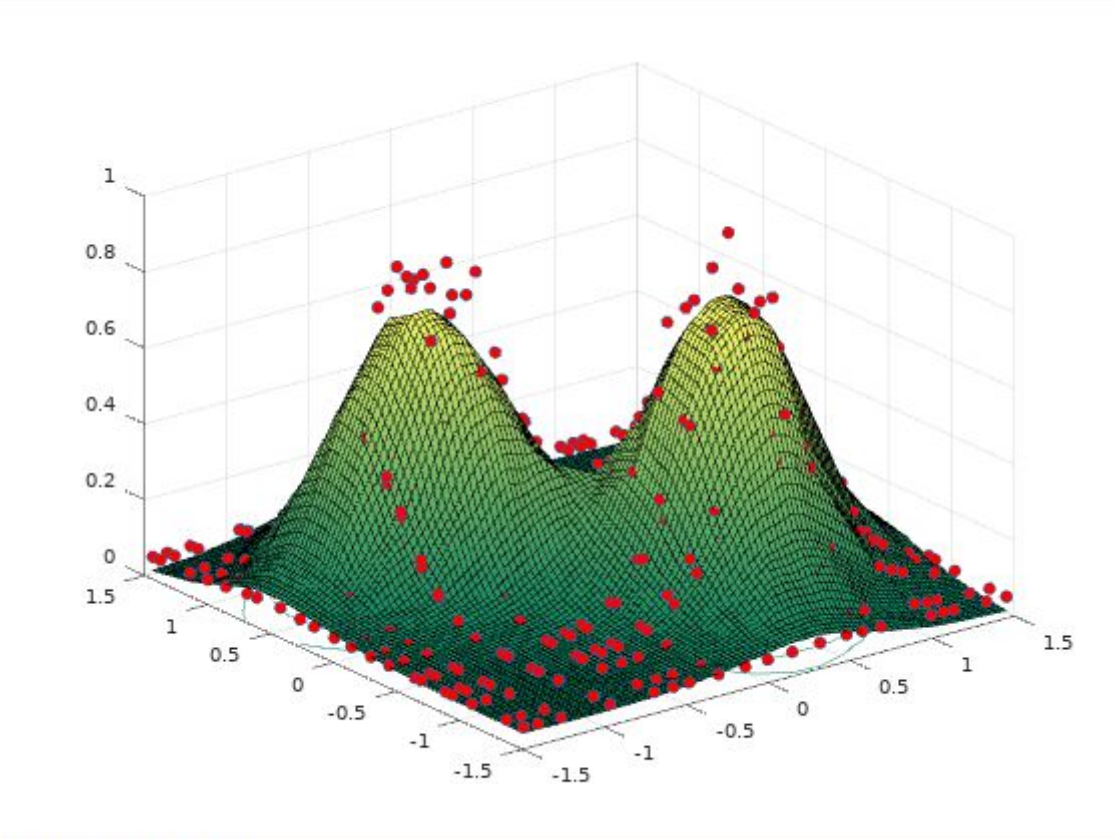
[Plano de error de generalización con 60% de aprendizaje]



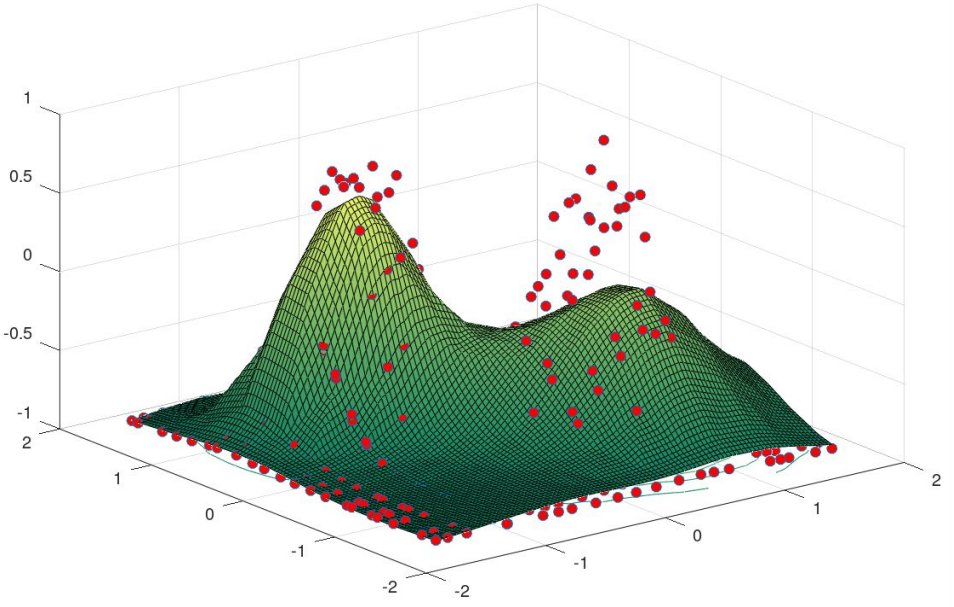
[Generalización con con 20% de aprendizaje]



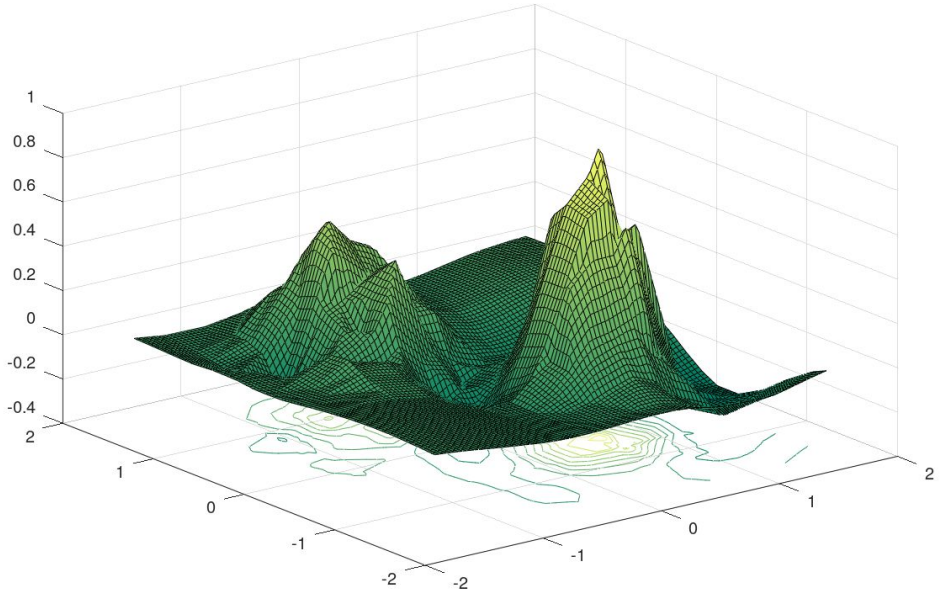
[Plano de error de generalización con 20% de aprendizaje]



Generalización 10%

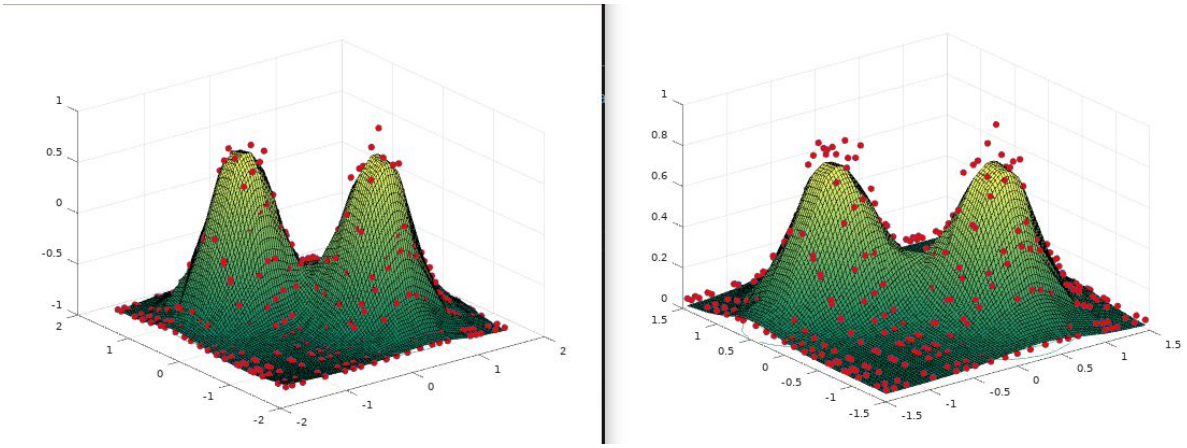


[Generalización con con 5% de aprendizaje]



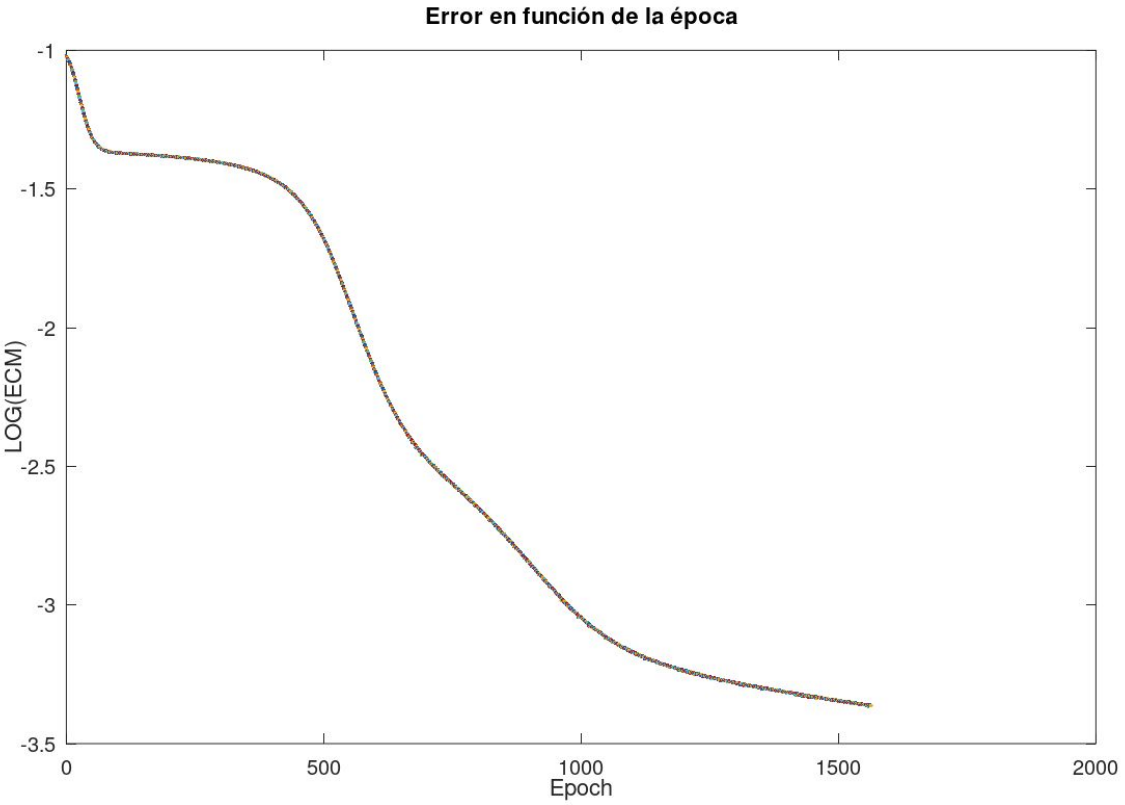
[Plano de error de generalización con 5% de aprendizaje]

Tangencial vs Exponencial

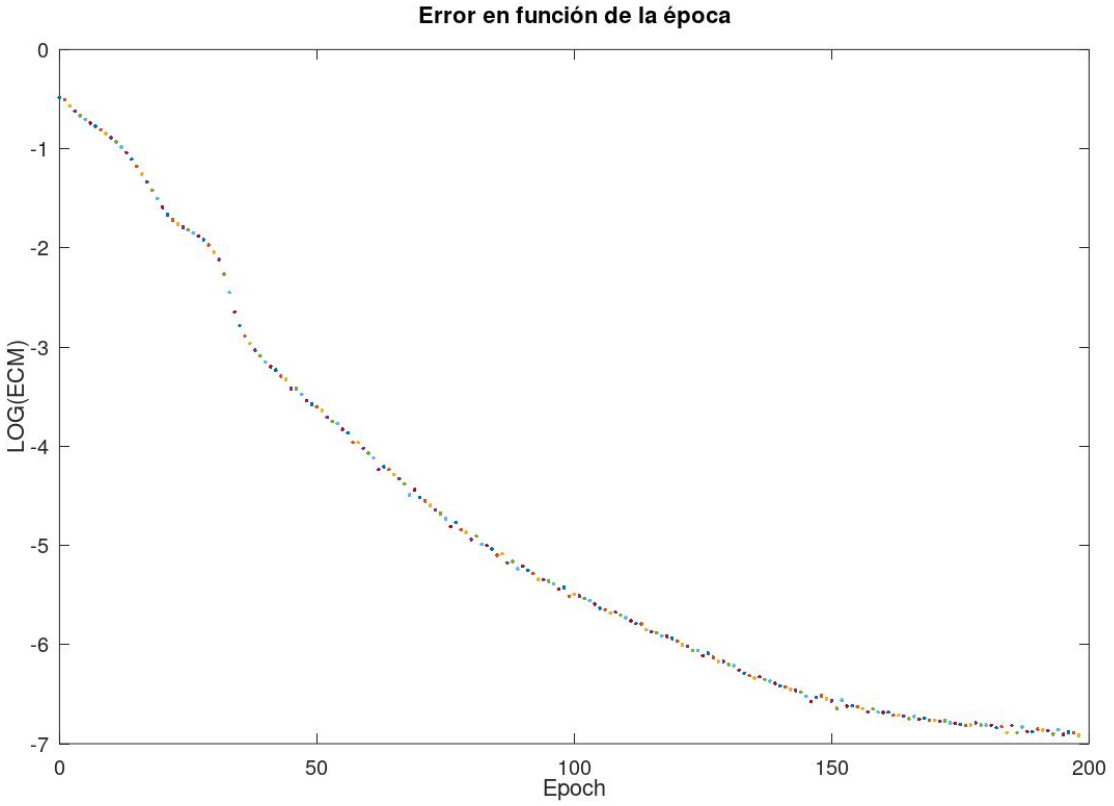


Comparación de gráficos generados (tangencial a la izquierda).

Beta



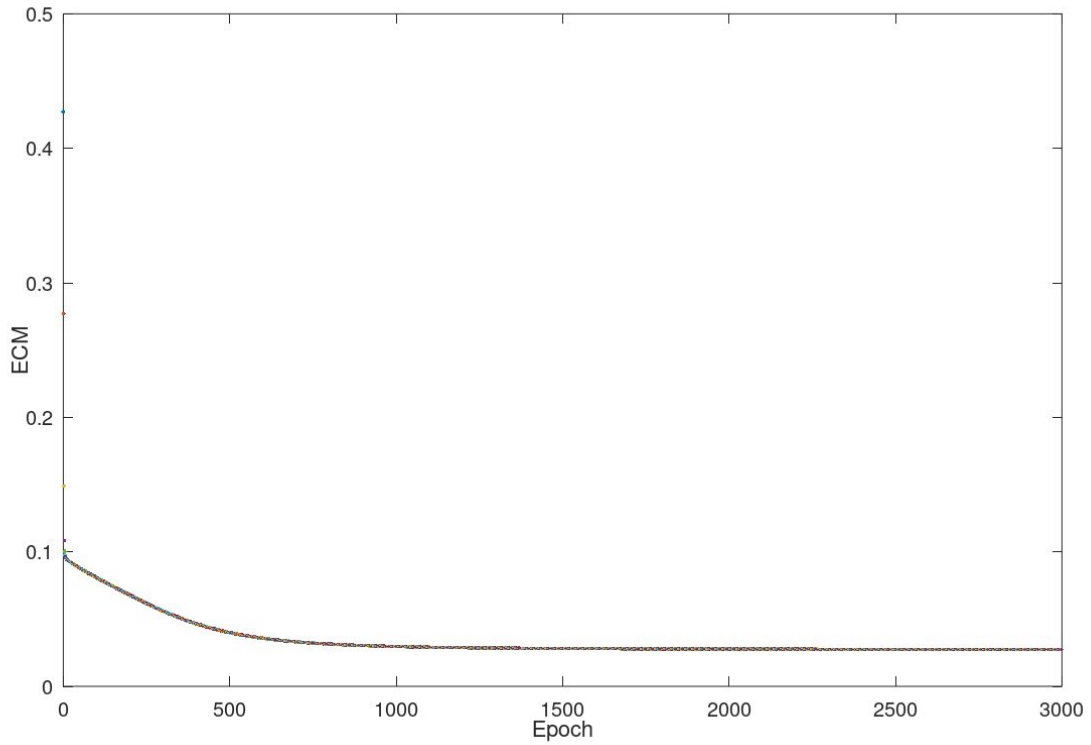
Beta = 0.5



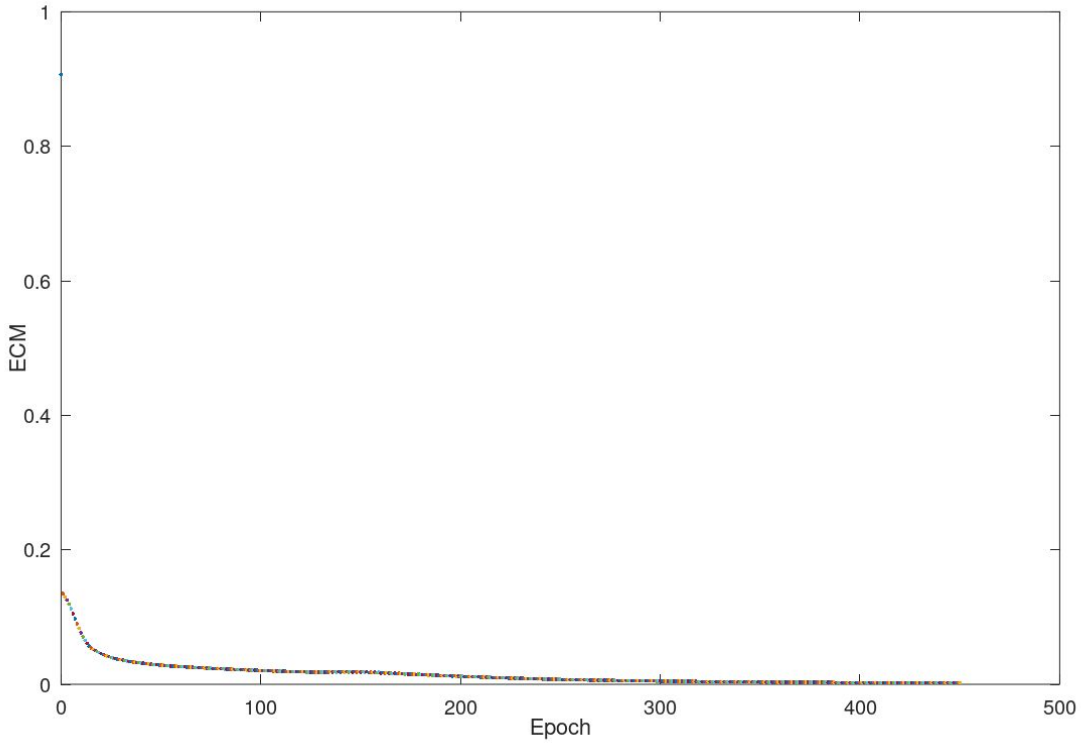
Beta = 1.5

Arquitectura

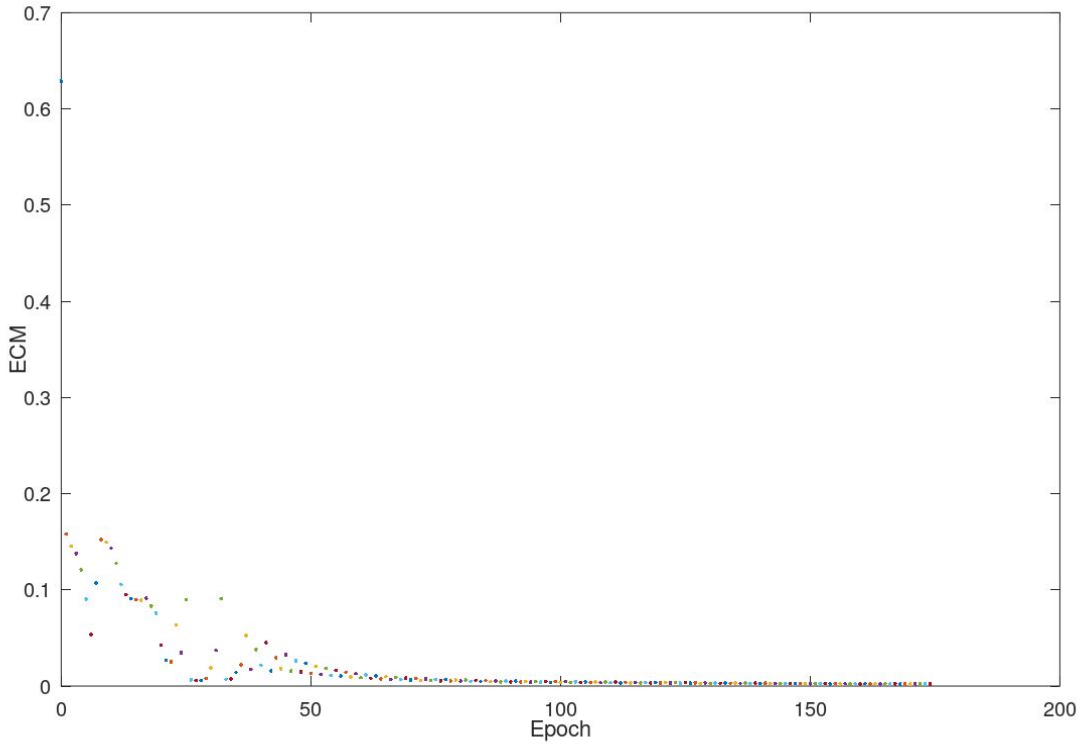
Cantidad de capas



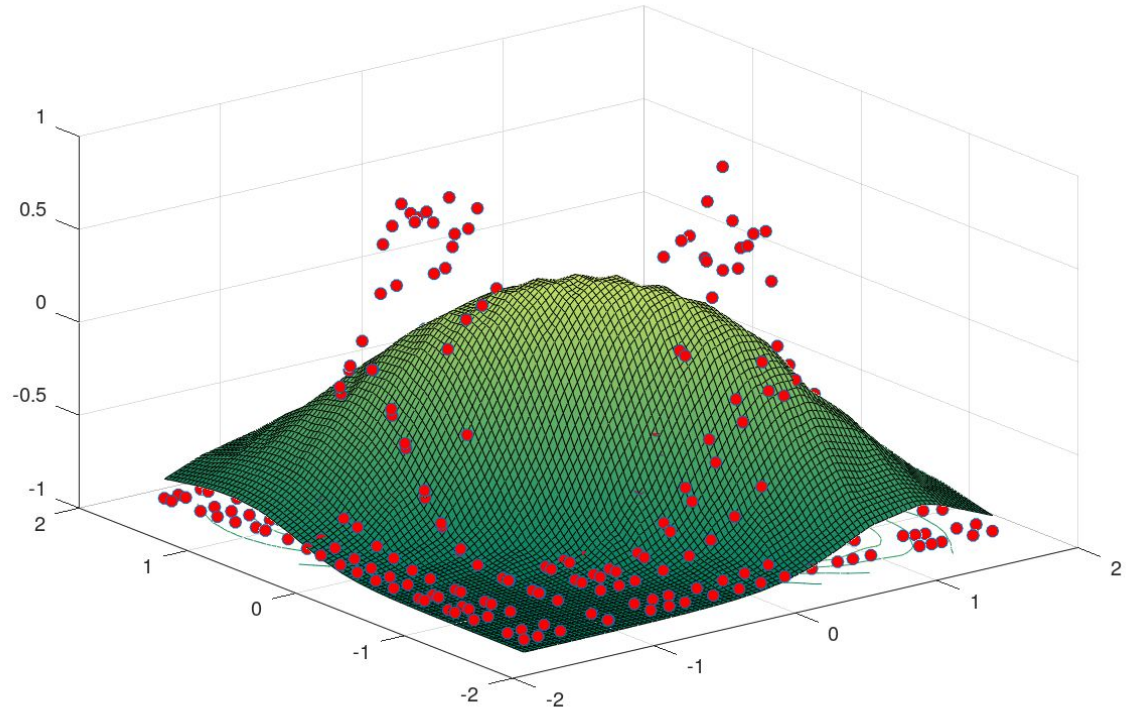
Arquitectura [50]



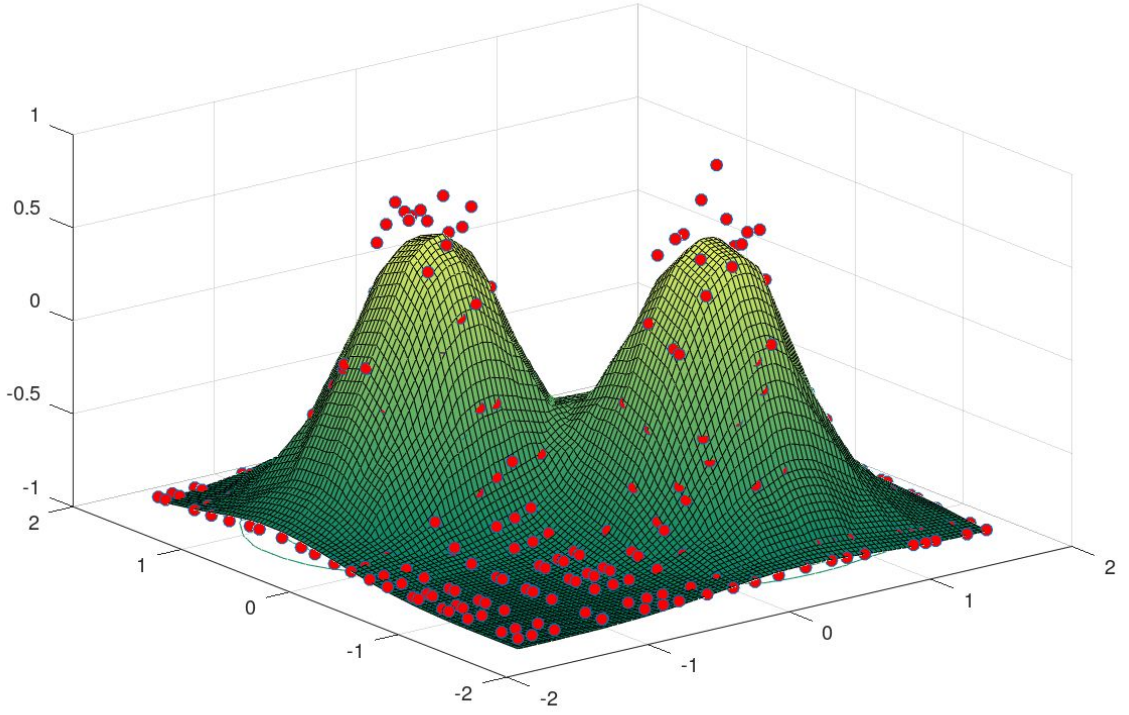
Arquitectura [50, 50, 50]



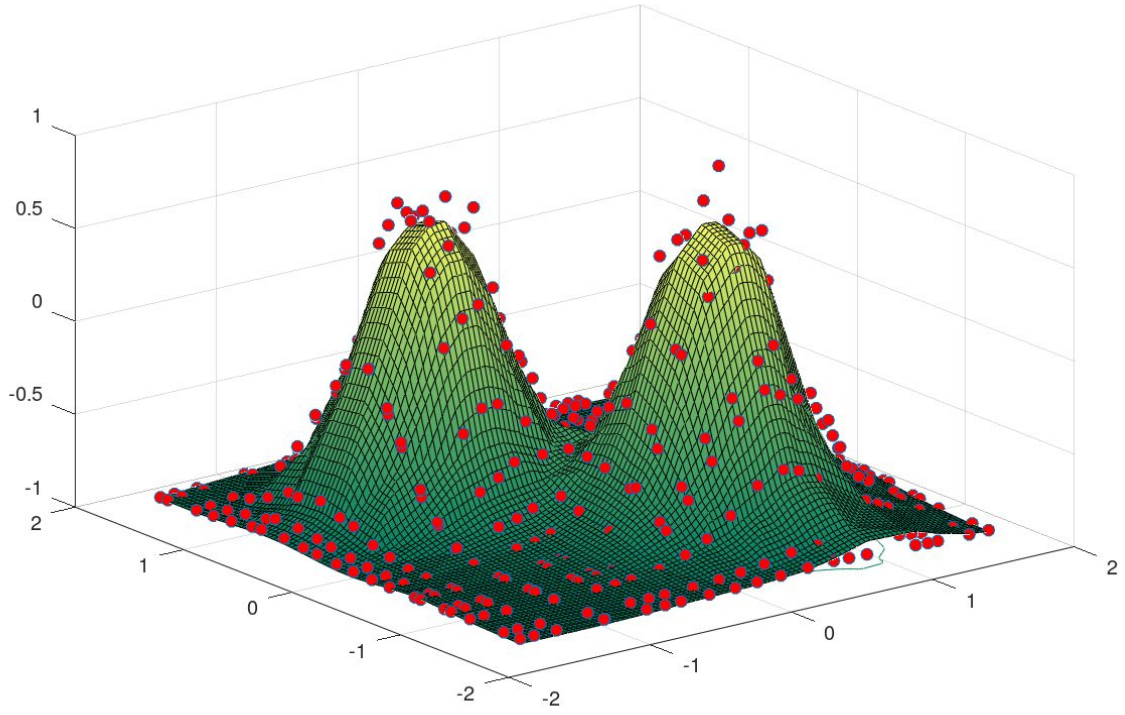
Arquitectura [50, 50, 50, 50, 50]



Arquitectura [50]

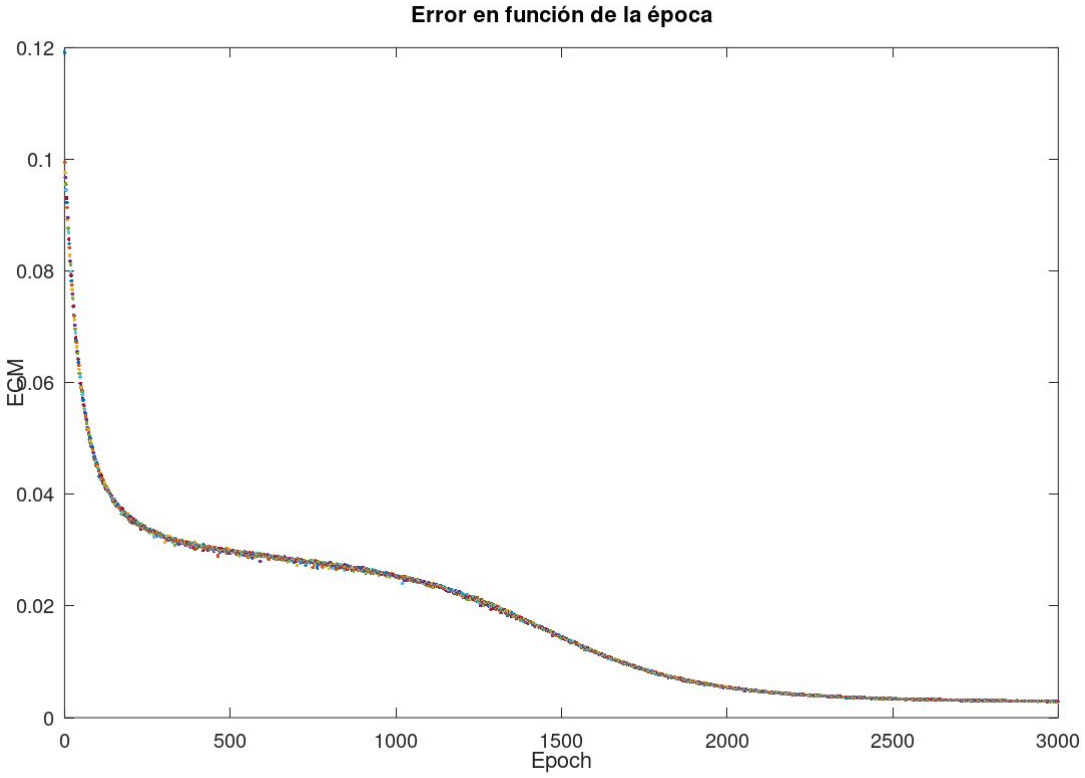


Arquitectura [50, 50, 50]

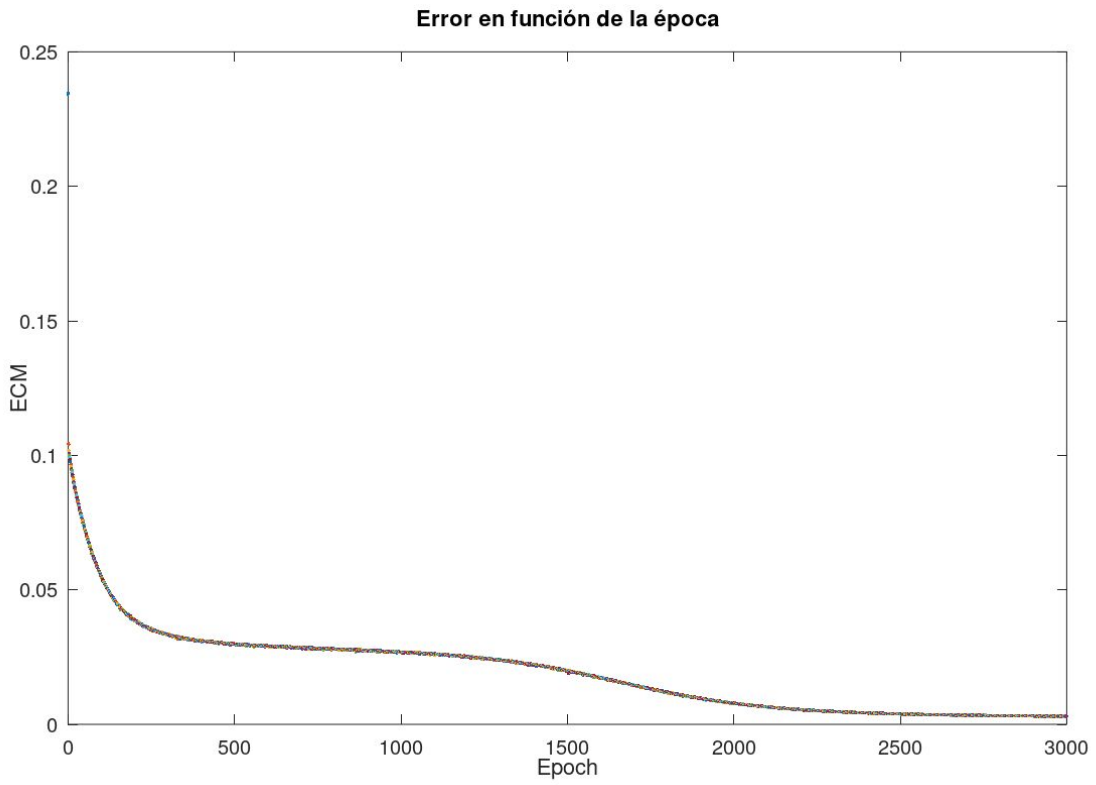


Arquitectura [50, 50, 50, 50, 50]

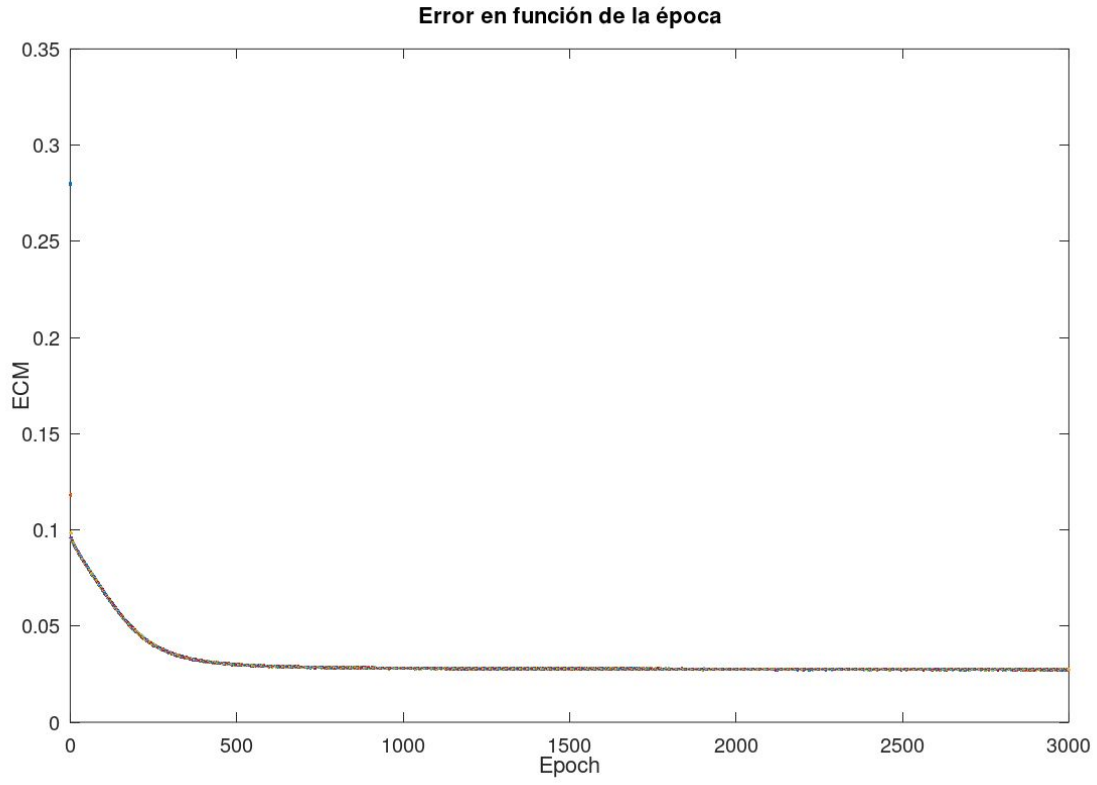
Una Capa



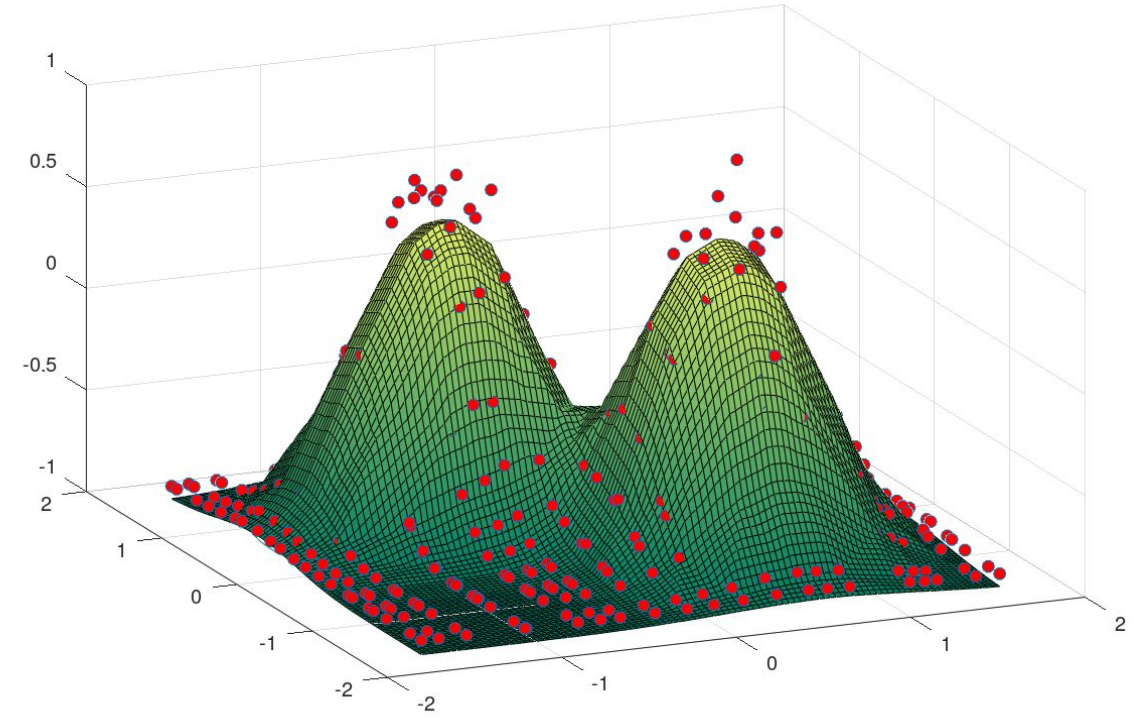
200 neuronas



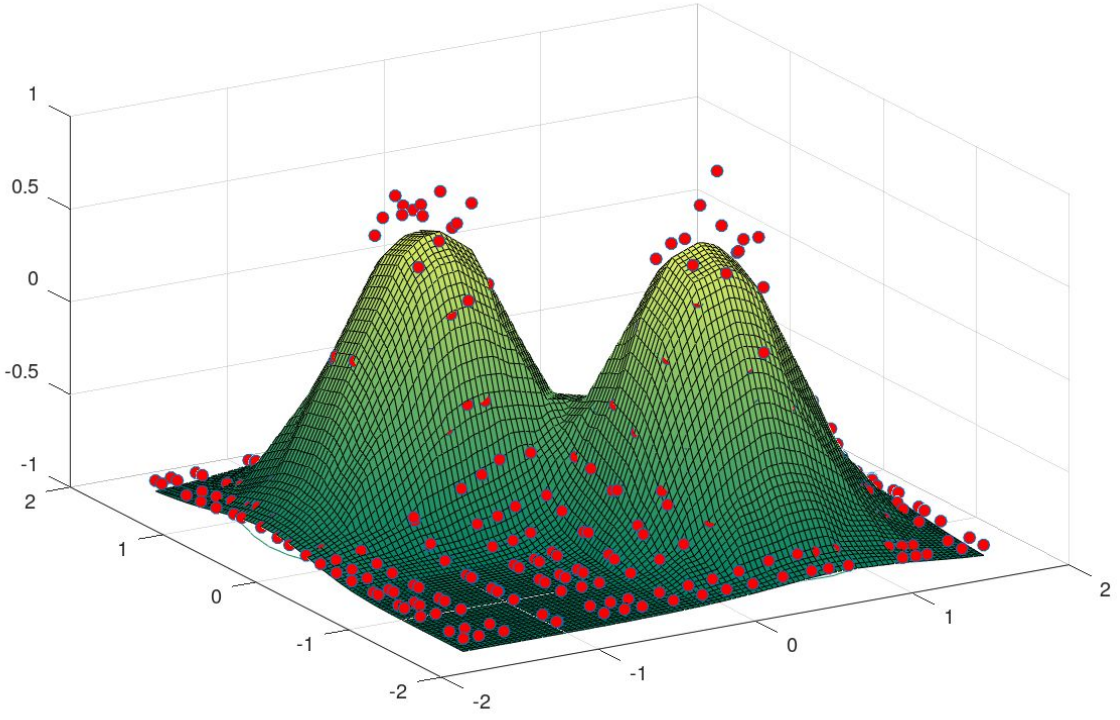
120 neuronas



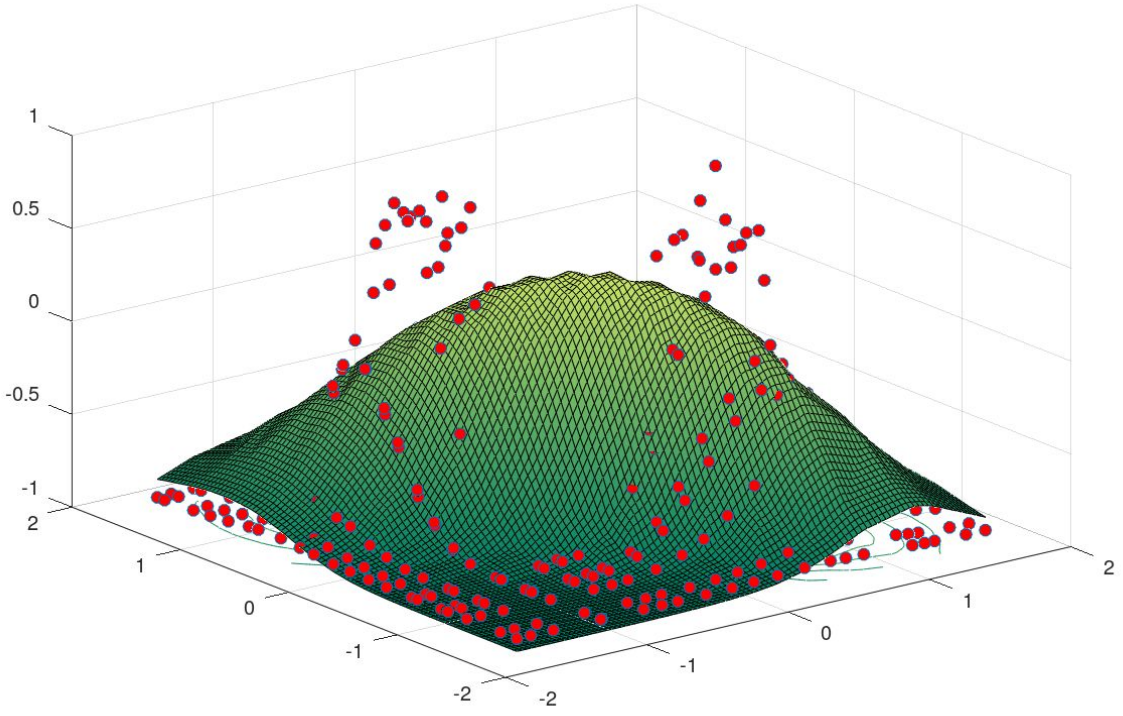
60 neuronas



200 neuronas



120 neurona



60 neuronas