INSTITUTO TECNOLÓGICO DE BUENOS AIRES
SISTEMAS DE INTELIGENCIA ARTIFICIAL
INGENIERÍA EN INFORMÁTICA

Redes Neuronales

Segundo trabajo práctico especial

Titular: Parpaglione, Cristina
Semestre: 2019 1C
Grupo: 5
Repositorio: https://bitbucket.org/itba/sia-2019-1c-05/src/master/neural-networks/
Entrega: 13 de mayo de 2019
Autores: Banfi, Micaela (57293)
Guzzetti, Clara (57100)
Ritorto, Bianca (57082)
Vidaurreta, Ignacio (57250)

1. Introducción

En este trabajo especial se resuelve el problema de aproximar un terreno a partir de mediciones de altura, latitud y longitud utilizando diferentes arquitecturas de redes neuronales multicapa y entrenandolas bajo el modelo de aprendizaje supervisado. Se analiza la aplicación de diversas técnicas de optimización como son *Momentum* y *Eta adaptativo*.

2. Implementación

2.1. Descripción

Se tiene un archivo de configuración llamado **network_parameters.bin** en el que se pueden modificar todos los parámetros de la red. Una vez modificado, se cuenta con un script de Matlab que permite correr el programa. Parado en la carpeta de neural-networks, se ejecuta

```
octave run.m
```

Se graficará en vivo el error de cada época, y al terminar la ejecución se verán tres gráficos: el del terreno original, el del terreno creado por la red junto con los puntos de testeo, y el del error.

2.2 Arquitecturas

Se utiliza un *cell array* de matrices para representar las diferentes capas de la red. Se denomina W y contiene en el índice i, la matriz de pesos que le genera la capa i-1 a la capa i. Por lo tanto, si la arquitectura fuera de [2, 8, 1]:

W[1] = matriz de 8x3

W[2] = matriz de 1x9

2.2.1 Arquitecturas de prueba

Para medir los efectos de las arquitecturas se evaluaron los entrenamientos a partir de modelos arbitrarios con diferente cantidad de capas y de unidades por capa que se encuentran en el apéndice. El modelo que mostró los mejores resultados fue el siguiente: [2, 8, 10, 1].

2.3 Pesos iniciales

Para los pesos iniciales se genera un vector de matrices correspondientes al valor de las aristas entre nodos de capas consecutivas. A la hora de inicializarlos, lo abordamos de dos maneras diferentes:

- *Aleatorio*

  Esto sirve para romper con la simetría y proporciona una precisión mucho mayor. En este método, los pesos se inicializan muy cerca de cero, pero al azar. Esto ayuda a que cada neurona ya no realice el mismo cálculo.

- *He-et-al*

  En este método, los pesos se inicializan teniendo en cuenta el tamaño de la capa anterior, que ayuda a alcanzar un mínimo global de la función de costos de manera más rápida y eficiente. Los pesos aún son aleatorios pero

difieren en el rango dependiendo del tamaño de la capa anterior de las neuronas. Esto proporciona una inicialización controlada, por lo tanto, el descenso del gradiente más rápido y más eficiente.

2.4 Funciones de activación

Se definen dos funciones con sus derivadas, basadas en las clases teóricas: sigmoidea exponencial y tangente hiperbólica. Su uso se debe a que el valor de su derivada guarda relación con su valor propio, además de que son continuas y diferenciables.

2.5 Red

La red consta de una matriz de pesos por capa y una arquitectura dada.

Para determinar la red se configura en base al tipo de entrenamiento, *batch* o *incremental*, el *learning rate*[1], la optimización del algoritmo de backpropagation, *Momentum* o *Eta adaptativo*, la función de activación[2] *g* utilizada y la arquitectura de nodos por capa empleada.

3. Aprendizaje

Dado que el dominio de los datos de entrada (-3 a 3) era distinto al dominio de la función de activación ([-1 1] o [0 1]) para poder lograr que la red neuronal aprenda correctamente se tuvo que normalizar los datos antes. Para normalizar los datos se usaron diferentes métodos: por máximos mínimos, por la norma y gaussiana, siendo la última la que tuvo mejores resultados. Se aplicó únicamente a los datos de entrenamiento dado que esta es la única etapa en la que se utilizan las funciones de activación, por lo que es la única etapa que presenta este desafío.

Se utiliza el algoritmo de backpropagation. Se calcula la diferencia entre los resultados esperados y los reales, y de esta manera se ajustan los pesos en las capas. Más específicamente se regresa a la capa de entrada, capa por capa, comparando la entrada de capa con el error de capa. Cada capa, sin importar qué tan profunda sea, contribuye al error de la siguiente capa; por lo tanto, podemos ajustar los pesos teniendo en cuenta las capas anteriores, es decir, yendo de atrás para adelante corrigiendo el peso. Este algoritmo de entrenamiento consiste en los siguientes pasos:

1. Por cada época[3] se ingresa un conjunto de patrones que se ordenan de forma aleatoria para evitar siempre aprender de los mismos, y en el mismo orden, que podría causar que la red *memorice* en vez de aprender.

2. Para cada patrón:

    a. Se realiza una propagación hacia adelante (forward-propagation) aplicando las derivadas de *g* y se obtiene el resultado. La diferencia entre éste y el esperado se almacena como error.

    b. El error es propagado hacia atrás (back-propagation), que resulta en un vector de ajustes para cada capa. Aquí existe una diferencia sutil entre la forma de actualizar pesos batch o incremental : en el

[1] Tasa que representa la rapidez con la que la red reacciona a los cambios; en la mayoría de los casos, cuanto más grande es la red, más pequeña debería ser, esto hace que el entrenamiento sea más lento pero más preciso. Define con qué rapidez una red actualiza sus parámetros.
[2] Funciones no lineales que trabajan en la suma del producto de entrada y peso, y el sesgo y transfiere el resultado a las neuronas de la siguiente capa. Se utiliza para asegurarse de que la representación en el espacio de entrada se asigna a un espacio diferente en la salida.
[3] Un forward pass y un backward de todos los ejemplos de entrenamiento.

primero se suma el delta weight de cada capa y se actualiza al terminar el batch, mientras que en el segundo se actualizan los pesos de las capas entre patrones.

3. Al finalizar una época, se calcula el error. El mismo se grafica a fines de ilustrar posibles mesetas en el aprendizaje.

4. Se actualiza el valor del *learning rate* y el *momentum*, si es que dicha funcionalidad se encuentra activa.

El resultado de cada época es un nuevo vector de matrices de pesos, y define una red neuronal entrenada. Se parte de una red anterior para entrenar una nueva. Se debe finalizar luego de lograr un error apreciablemente pequeño habiendo pasado por un mínimo de épocas, o superada una cantidad arbitraria de épocas.

3.1 Optimizaciones en el algoritmo de backpropagation

3.2.1 Eta adaptativo

Se implementó en el desarrollo del algoritmo de backpropagation una mejora denominada como eta adaptativo. Esta mejora intenta "premiar" a la red, aumentar el eta en uno cuando el error que calcula es menor comparado al error de la época anterior, y "castigarla" cuando es un error mayor. Esto se realiza cambiando dinámicamente el valor de eta. Se implementó con un vector de entrada con 3 parámetros (el vector nulo representando la no utilización de esta mejora). El primer parámetro es α, el segundo es β y el tercer es la cantidad de veces consecutivas que tiene que bajar el error para aplicarle el eta adaptativo. Cada vez que se aplica α, se pone *momentum* en 0.9, y cada vez que se tiene que aplicar β, en 0.

3.2.2 Momentum

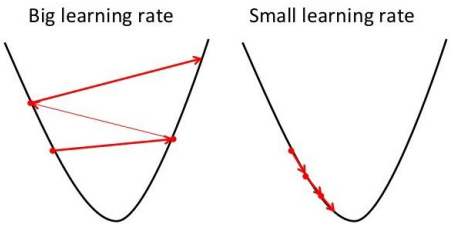
Se implementó la mejora al algoritmo *momentum*. Este se ingresa por parámetro y sus valores pueden variar de 0-1. Momentum es el análogo de aplicar inercia sobre una caminata de un terreno. Se aplica de esta forma (W siendo el vector de las matrices de pesos):

```
W{i} += delta_W{i} + momentum * last_delta_W{i}
```

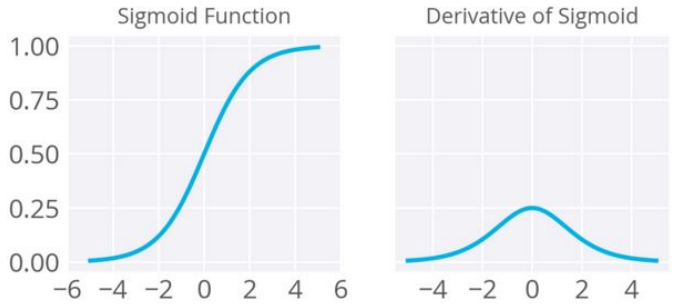
4. Conclusiones

4.1 Comportamiento

- **Variaciones según los hiperparámetros:** El comportamiento de NN cambia con respecto a la cantidad de datos, y también depende de los métodos de entrenamiento, el número de épocas y la arquitectura de la red.
- **Efecto del Learning Rate:** Se sabe que *learning rates* altos pueden prevenir la convergencia hacia los mínimos al rondar alrededor, pero al mismo tiempo, si la velocidad de aprendizaje es demasiado lenta, es posible que no podamos converger en los mínimos a medida que avanzamos ya que se toman muy pequeños pasos hacia los mismos.



- **Implicancias de las funciones de activación:** Las funciones sigmoide y tanh son curvas en sus extremos, lo que hace que los gradientes en esos puntos sean bastante bajos porque incluso cuando el valor antes de aplicar la activación sigmoide aumenta considerablemente, el valor después de la aplicación no lo hace y, por lo tanto, incluso el gradiente no aumenta mucho en el impacto del aprendizaje. De todas maneras, Sigmoid y Tanh pueden ser buenas opciones para redes poco profundas (máx. 2 capas).



4.2 Análisis de resultados

A continuación se intentan exponer los aspectos más significativos del problema y sus resultados.

4.2.1 Sigmoide vs. Tangente hiperbólica

La función que mejor resulta al terreno que se analizó fue la de tangente hiperbólica. El error que alcanza es significativamente menor.

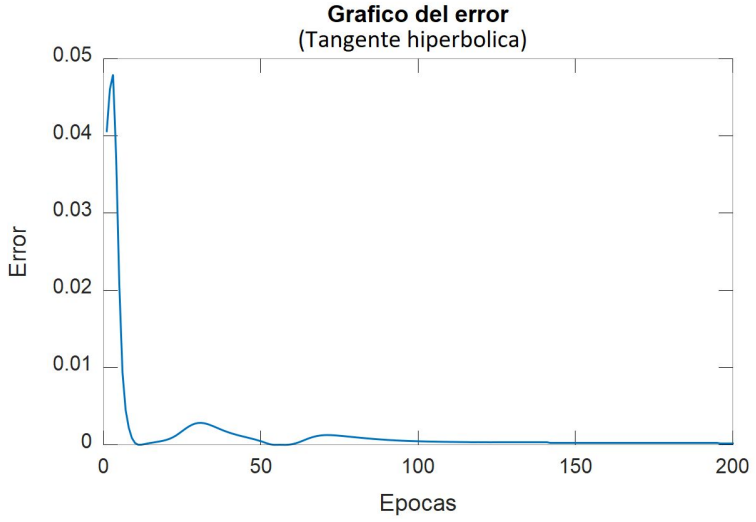


Gráfico con función de activación tangente hiperbólica

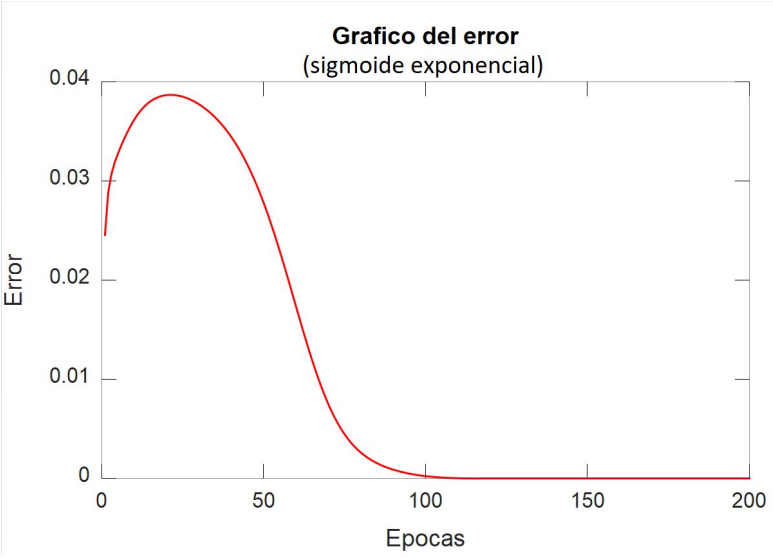


Gráfico con función de activación sigmoidea

4.2.1 Incremental vs. Batch

Se notó que en este caso, el aprendizaje batch tardaba mucho más en llegar al delta de error deseado que incremental. Se llegaba al error, pero en muchas más épocas.

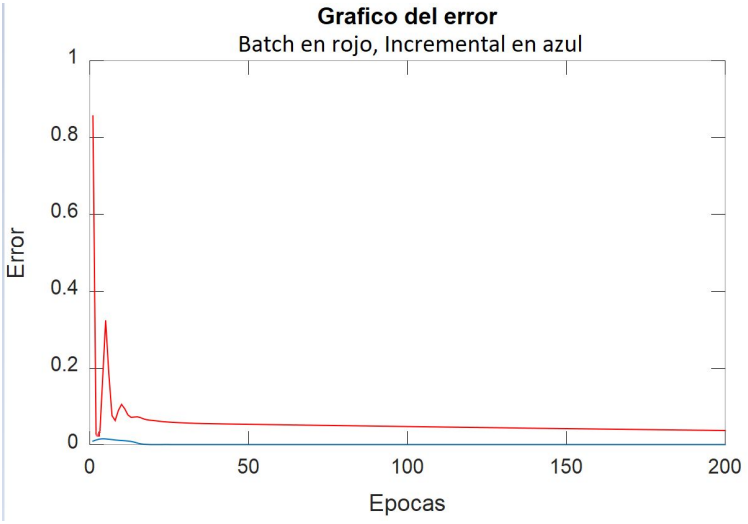


Gráfico error con batch(rojo) e incremental(azul)

4.2.2 Eta vs eta adaptativo

No se pudieron encontrar valores para el eta adaptativo que mejoraran la performance de la NN, y por eso se decidió no utilizar esta mejora en el entrenamiento de la red. A continuación, se muestran gráficos de la variación del eta adaptativo y el error.

Eta: α=1.4, β=0.5, Tercer parámetro=5

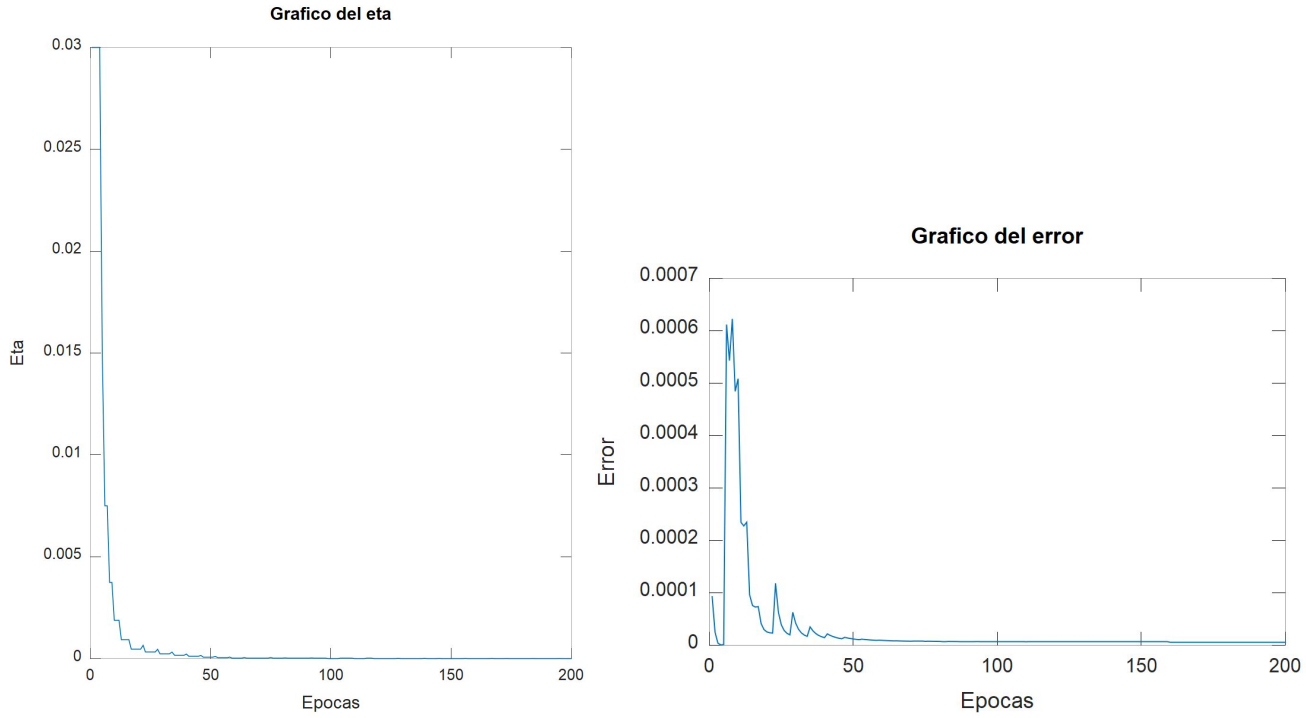


Gráfico del eta adaptativo y el error

Eta: α=0.5, β= 1.5, Tercer parámetro= 3

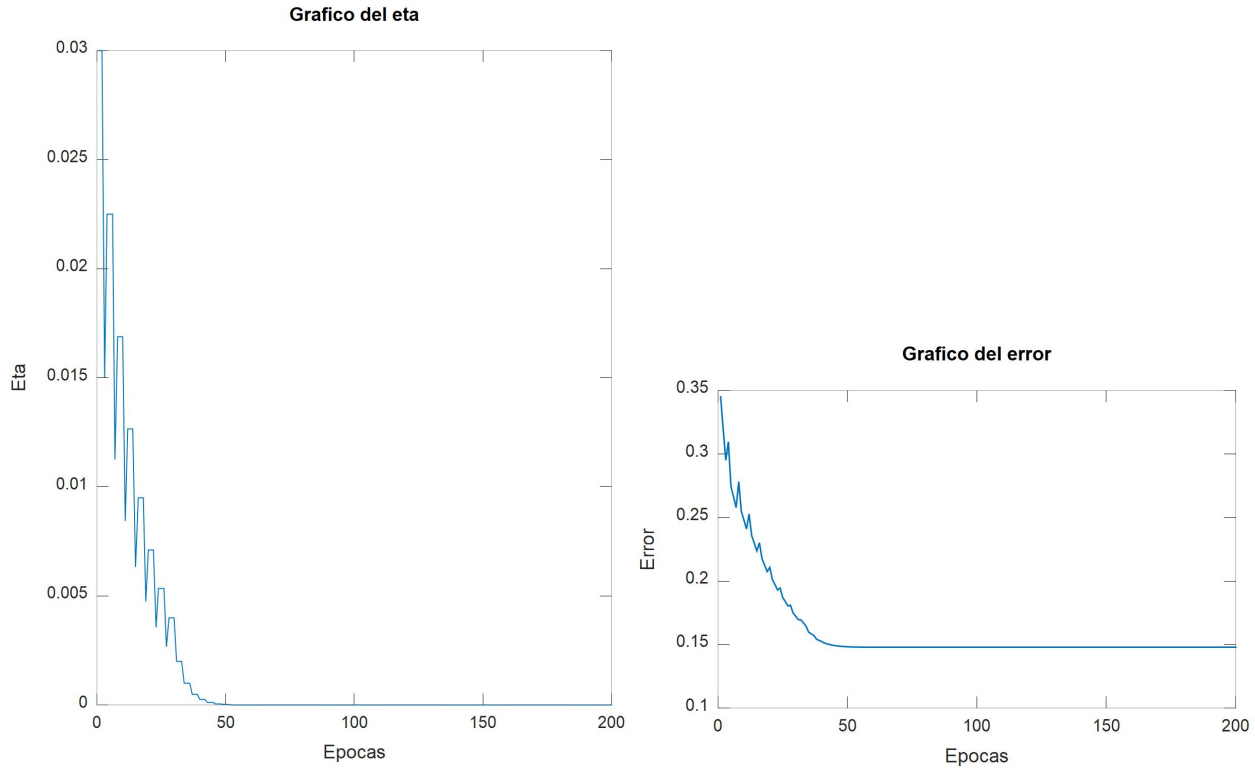
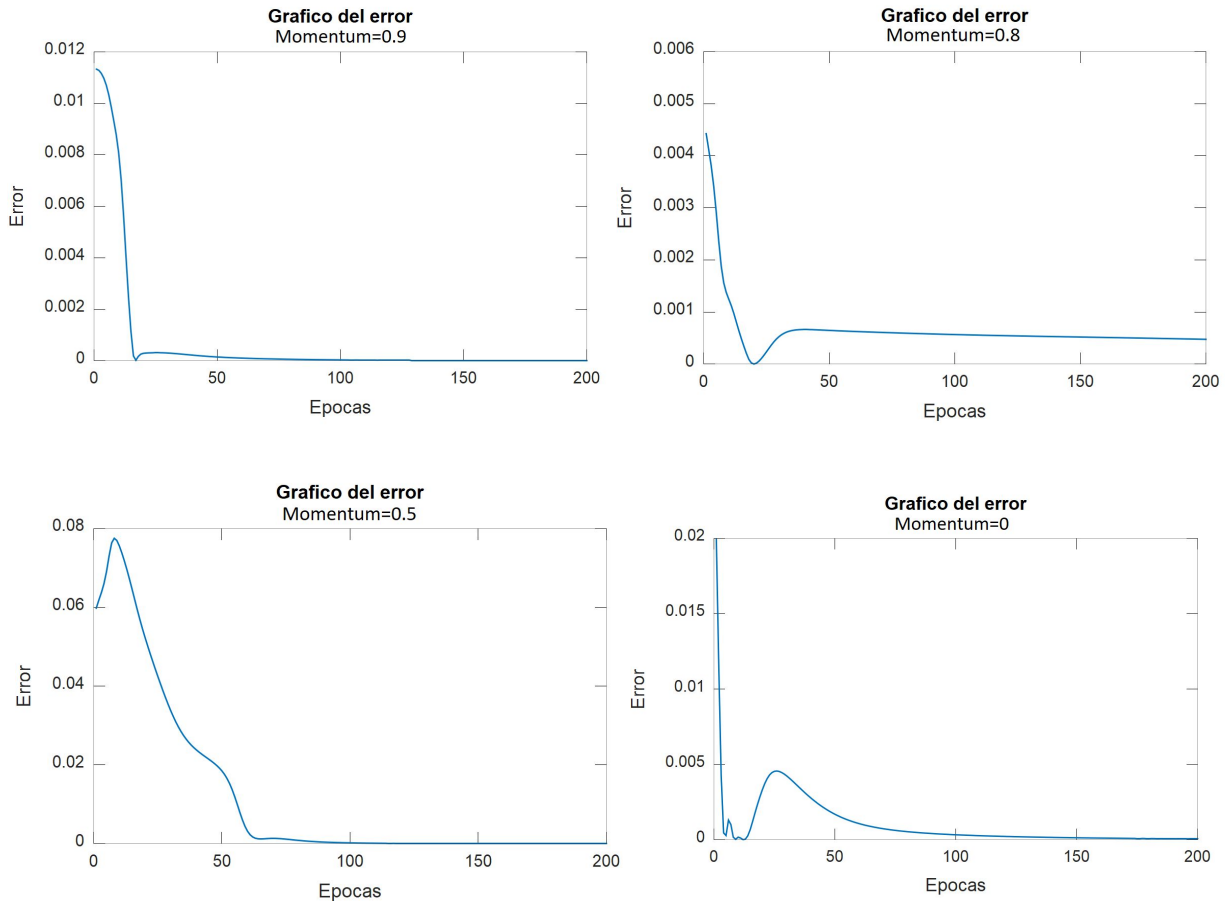


Gráfico del eta adaptativo y el error

4.2.3 Momentum vs sin momentum

Se observó un mejor y más rápido aprendizaje de la red cuando se utilizan valores variantes del momentum entre 0.8 y 0.9, dependiendo de la arquitectura. A continuación se muestran gráficos de error con diferentes momentums.



4.3 Conclusión parámetros

Se pudo concluir analizando las cuestiones mencionadas en el inciso 4 y la tabla de testeos del apéndice, que los mejores parámetros para obtener un error de 0.00001 rápidamente son: aprendizaje incremental, función de activación tanh, pesos inicializados aleatoriamente, normalización Gaussiana, eta 0.03 sin adaptativo, y momentum de 0.9.

4.4 Error de la red luego de ser entrenada

Con los valores reservados para hacer el testing de la red se calculó el error que, luego de ser entrenada, la red tenía para graficar estos valores que nunca antes había visto. Con los mejores parámetros que se encontraron, la red produjo un error de 4.7799e-04. Se muestra la salida del programa en el apéndice.

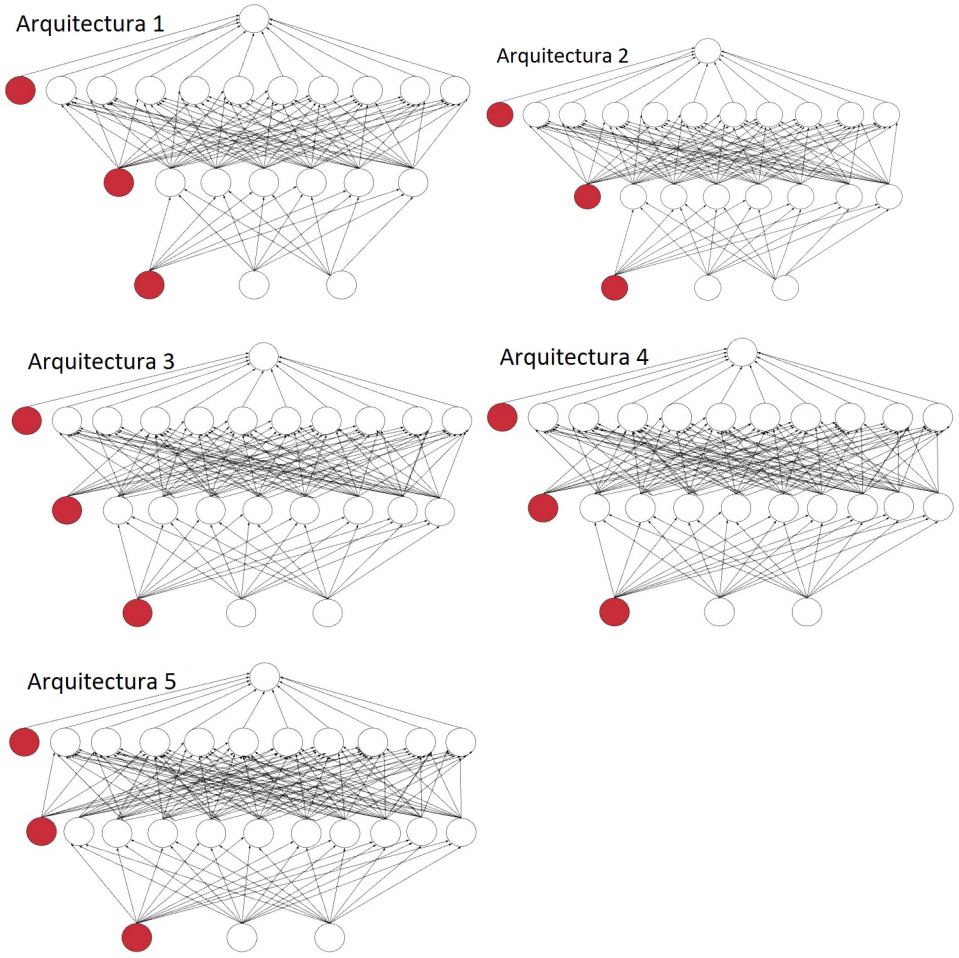
4.5 Mejoras posibles

- Al no manejar muy bien el lenguaje Matlab, muchas cosas se hicieron utilizando iteraciones *for*, que luego se investigó, realentizan la ejecución del programa. Se podría investigar más a fondo si hay cosas que pueden cambiarse y realizarse con funciones de Matlab ya preprogramadas, y así mejorar la performance del código.

- Conseguir valores correctos de *eta adaptativo* para acelerar el proceso de aprendizaje de la red. No se pudieron encontrar estos valores, pero quizás probando más combinaciones se podrían haber encontrado.
- Más pruebas con mejor espectro de parámetros.

5. Apéndice

5.1. Gráficos de diferentes arquitecturas



5.2. Tabla de diferentes arquitecturas

Se utilizaron para todos los testeos: aprendizaje incremental, función de activación tanh, pesos inicializados aleatoriamente, y normalización Gaussiana y se midió el error alcanzado en 200 épocas.

Terrain	Architecture	eta	momentum	Error (200 épocas)
Terrain05	2 6 10 1	0.08	0.9	1.6275e-4
Terrain05	2 7 10 1	0.08	0.9	6,12E-05
Terrain05	2 8 10 1	0.08	0.9	2,08E-06
Terrain05	2 9 10 1	0.08	0.9	3,34E-05
Terrain05	2 10 10 1	0.08	0.9	1,89E-05
Terrain05	2 6 10 1	1.4, 0.85, 3	0.9	2,42E-05
Terrain05	2 4 8 1	0.03	0.9	8,13E-07
Terrain07	2 6 10 1	0.08	0.9	2,50E-05

Terrain07	2 7 10 1	0.08	0.9	5,30E-04
Terrain07	2 8 10 1	0.08	0.9	0,038
Terrain07	2 9 10 1	0.08	0.9	1,40E-05
Terrain07	2 10 10 1	0.08	0.9	1,53E-05
Terrain14	2 6 10 1	0.08	0.9	0,0017409
Terrain14	2 7 10 1	0.08	0.9	0,0018283
Terrain14	2 8 10 1	0.08	0.9	0,0036958
Terrain14	2 9 10 1	0.08	0.9	0,0045899
Terrain14	2 10 10 1	0.08	0.9	8,31E-04

5.3 Salida del programa con los mejores parámetros

Terrain05. Arquitectura [2, 8, 10, 1]. En rojo, “o” los puntos de testeo. Error de generalización: 4.7799e-04.