Sistemas de Inteligencia Artificial

Trabajo Práctico Especial II

Grupo 4

Docentes

Parpaglione, María Cristina

Luciani, Ignacio José

Alumnos

Badi, Leonel

Farré, Lucas Andrés

Gómez, Jorge Exequiel

1. Descripción

1.1. Objetivo

Implementar una red neuronal multicapa con aprendizaje supervisado con la cuál se pueda estimar la función:

$$y(x) = sinh(x)\ cos(x^2),\ x \in [-2,\ 2]$$

1.2. Implementación

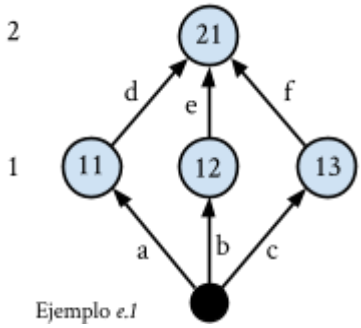
En este apartado explicaremos detalladamente cómo implementamos el algoritmo presentado en clase mediante el uso de matrices. Para esto, presentaremos una red neuronal de ejemplo y mostraremos la manera en que se realizan las distintas operaciones matriciales.

1.2.1. Lenguaje de programación

Para la implementación del algoritmo se decidió utilizar el lenguaje *GNU Octave*, debido a que ofrece la capacidad de cálculo con matrices y vectores, y de realizar gráficos mediante *gnuplot*.

1.2.2. Pesos y umbrales

Para representar los pesos y umbrales decidimos crear una matriz w_i por cada capa *i*. En la primera fila de la matriz se encuentran los umbrales de las neuronas, y en las filas siguientes, los pesos de las respectivas conexiones de cada neurona con la capa anterior. Es decir que, si por ejemplo la red fuese la siguiente:



Ejemplo *e.1*

, w_1 resultaría: $\begin{pmatrix} u_{11} & u_{12} & u_{13} \\ a & b & c \end{pmatrix}$,

y w_2: $\begin{pmatrix} u_{21} \\ d \\ e \\ f \end{pmatrix}$, donde *a*, *b*, *c*, *d*, *e* y *f* son los pesos y los u_{ij}, los umbrales.

1.2.3. Salidas

El cálculo de las salidas de cada capa, teniendo en cuenta lo especificado en el punto 1.2.2, resulta ser simplemente la multiplicación matricial de las unidades de entrada de cada capa con las matrices w_i.

Para el ejemplo *e.1*, las salidas de la capa 1 resultan:

$$h_1 = [-1,\ \xi_1] \cdot w_1 = [\ h_{11},\ h_{12},\ h_{13}\ ]$$
$$V_1 = g(h_1) = [\ g(h_{11}),\ g(h_{12}),\ g(h_{13})\ ],$$

y para la capa 2:

$$h_2 = [-1,\ V_1] \cdot w_2 = [-1,\ g(h_{11}),\ g(h_{12}),\ g(h_{13})\ ] \cdot w\_2 = [\ h_{21}]$$
$$V_2 = g(h_2) = [\ g(h_{21})\ ].$$

1.2.3. Deltas (δ)

Para calcular δ_i de la capa de salida i , multiplicamos elemento a elemento $g'(h_i)$ por la diferencia entre la salida esperada S_i y la salida obtenida h_i .

Siguiendo con el ejemplo *e.1*, δ_2 entonces resulta:

$$\delta_2 = g'(h_2) \cdot* (S_2 - h_2) = [g'(h_{21})] \cdot* [S_{21} - h_2)] = [\delta_{21}] \cdot$$

Los δ_j de las capas ocultas j , en cambio, se calculan multiplicando elemento a elemento $g(h_j)$ por la traspuesta de la matriz resultante de multiplicar los w_i por el δ_i de la capa superior *i*, pero habiendo eliminado previamente la primera fila de umbrales de w_i .

Volviendo al ejemplo *e.1*, δ_1 resulta:

$$\delta_1 = g'(h_1) \cdot* (w_2 \cdot \delta_2)^T = [\ g'(h_{11}),\ g'(h_{12}),\ g'(h_{13})\ ] \cdot* [d\ \delta_{21},\ e\ \delta_{21},\ f\ \delta_{21}]$$
$$\delta_1 = [\ g'(h_{11})\ d\ \delta_{21},\ g'(h_{12})\ e\ \delta_{21},\ g'(h_{13})\ f\ \delta_{21}\ ] = [\delta_{11},\ \delta_{12},\ \delta_{13}]$$

1.2.4. Actualización de los pesos

Una vez calculados los δ_i , procedemos a calcular los Δw_i . Para esto, multiplicaremos elemento a columna aprovechando la función de *broadcasting* automático de *Octave*, esto es, al realizar la multiplicación elemento a elemento de vectores fila por vectores columna, como las dimensiones no se corresponden, *Octave* automáticamente hace la multiplicación de cada elemento del vector fila por el vector columna, que es justamente lo que queremos que suceda al realizarse la operación:

$$\Delta w_i = \eta\ \delta_i \cdot* \ [-1,\ V_j]^T$$

Para el ejemplo *e.1* obtenemos:

$$\Delta w_1 = \eta\ \delta_1 \cdot* \ [-1,\ \xi_1]^T = [\eta\delta_{11},\ \eta\delta_{12},\ \eta\delta_{13}] \cdot* [-1,\ \xi_1]^T$$
$$\Delta w_1 = [-\eta\delta_{11},\ -\eta\delta_{12},\ -\eta\delta_{13};\ \eta\delta_{11}\xi_1,\ \eta\delta_{12}\xi_1,\ \eta\delta_{13}\xi_1]$$
$$\Delta w_2 = \eta\ \delta_2 \cdot* [-1,\ V_1]^T = [\eta\delta_{21}] \cdot* [-1,\ g(h_{11}),\ g(h_{12}),\ g(h_{13})\ ]^T$$
$$\Delta w_2 = [-\eta\delta_{21};\ \eta\delta_{21}g(h_{11});\ \eta\delta_{21}g(h_{12});\ \eta\delta_{21}g(h_{13})\ ]$$

Finalmente actualizamos los pesos, sumando a los pesos anteriores el Δw_i y el $\Delta w_{i\ old}$ multiplicado por el coeficiente de *momentum*, α .

En el ejemplo *e.1*,

$$w_{1\ new} = w_{1\ old} + \Delta w_1 + \alpha\Delta w_{1\ old}$$
$$w_{2\ new} = w_{2\ old} + \Delta w_2 + \alpha\Delta w_{2\ old}$$

1.2.4. Parámetros

1.2.4.1 Neuronas en la capa oculta

Para la elección de la cantidad de neuronas en la capa oculta se decidió implementar una función que ejecuta el algoritmo con cantidades de 1 a 10, y diferente número de épocas. Luego se realiza una comparación de gráficos entre la función y los valores de retorno del conjunto de testeo.

1.2.4.2 Momentum

Para el momentum, se implementó la propuesta clásica utilizando un α que se recibe como parámetro. Para desactivar esta mejora, basta con pasarle cero como parámetro.

1.2.4.3 Parámetros adaptativos

Para la implementación del η adaptativo se necesitó tomar la decisión de los valores de *a* (parámetro de incremento constante) y de *b* (parámetro de decremento geométrico). Para ello, se decidió implementar una función de testeo que ejecuta el algoritmo con diferentes posibilidades de *a* y *b*. Para cada posible combinación de parámetros, se corre el algoritmo de aprendizaje una cierta cantidad de veces, se calcula el promedio de los errores de los mismos y se guardan los datos en un archivo. En el mismo se guardan el valor inicial de η, a, b, incrementos y decrementos de eta, iteraciones y error promedio.

En nuestra implementación decidimos no corregir la matriz de w cada vez que el error aumenta, de esta manera pudimos superar ciertos mínimos locales.

Si se desea desactivar esta funcionalidad, solo basta con poner *a* y *b* en cero.

2. Resultados

2.1. Detalles numéricos

Los detalles numéricos de los resultados obtenidos se encuentran en la tabla del punto A.1. del Anexo.

2.2. Análisis de los resultados

2.2.1 Variación en la cantidad de neuronas

Dada la naturaleza de la función a aproximar, necesitamos cuatro neuronas en la capa oculta para poder representarla con un error cuadrático medio cercano a *0,01*. Esto se debe a la baja cantidad de máximos y mínimos que se presentan en el intervalo de trabajo (6, incluyendo los extremos).

Una observación que cabe destacar, es el aumento de la *flexibilidad* de la red al aumentar la cantidad de neuronas. Cada vez que la misma se aumenta en una unidad, se puede apreciar la capacidad de la red de realizar un cambio en su dirección (ver gráficos de la sección *A.4*).

Como se puede ver en el gráfico a partir de la utilización de 3 neuronas, ya se consigue una aproximación bastante acertada de la función.

2.2.2 Impacto del *momentum*

La implementación del momentum permite poder reducir el error cuadrático medio en un menor número de iteraciones comparado al no implementarlo. Esto se puede ver en el gráfico de la sección *A.3*, donde el error cuadrático medio se reduce a *0,07* en 4 iteraciones utilizando el mismo. En cambio, sin utilizarlo, llevó 12 iteraciones. Ambas redes fueron inicializadas con las mismas matrices de pesos.

2.2.3 Elección de patrones de aprendizaje

Al momento de la selección de patrones de aprendizaje, se decidió realizar pruebas con tres distribuciones diferentes. El primero de ellos con un peso en la parte de los extremos, es decir, aumentando la cantidad de patrones en los lugares del máximo y mínimo de la función, ya que en esa zona se produce un error mayor. Luego se decidió tomar una distribución reducida de patrones para analizar si la diferencia de error obtenida con el anterior era relativamente pequeña y así poder hacer más ágil el cálculo de cada época. Y por último se realizaron pruebas utilizando una distribución constante de patrones incrementados en *0,02*.

Se pudo observar, como se muestra en la tabla de resultados *A.1*, que al momento de reducir la cantidad de patrones, el error es mucho mayor al obtenido en cualquiera de las otras dos distribuciones. Como además el tiempo de aprendizaje para este ejercicio no se toma en cuenta, decidimos tomar la distribución que nos generó menor error, la uniforme.

2.2.4 Problema de los extremos

Luego de realizar diferentes pruebas con distintos valores de los parámetros con una única capa oculta, no pudimos lograr que la red aprenda correctamente tanto los primeros valores de f(x) con *x* cercanos a *-2*, como para los últimos, con *x* cercanos a *2*. Como todo el algoritmo fue implementado para una sola capa oculta, decidimos agregar una capa más al código para ver si el problema se solucionaba. Lamentablemente, los errores en los extremos siguieron resultando mayores que en el resto de la función. Por esa razón, decidimos volver el código a la versión anterior para que el mismo resulte más simple.

2.2.5 η adaptativo

Luego de analizar los resultados obtenidos en la tabla *A.2*, y realizar las comparaciones de errores con las diferentes posibilidades de parámetros de incremento y de decremento de η, pudimos observar que se puede llegar a un error menor utilizando esta mejora.

Como a nosotros no nos marca una diferencia la cantidad de incrementos y decrementos del mismo, ya que no descartamos la matriz *w* para escapar de posibles mínimos, decidimos elegir la combinación que nos devolvió un error cuadrático medio menor. Es importante aclarar que, como los pesos son inicializados al azar, cualquiera de las primeras combinaciones son una buena elección de parámetros para la utilización de esta mejora.

3. Conclusiones

El análisis y entendimiento del problema a resolver cumple un papel fundamental en el aprendizaje de la red, ya que si bien la estructura es similar para todos, los parámetros de aprendizaje pueden variar drásticamente. Por ejemplo, la cantidad de neuronas a colocar en la capa oculta varía con la cantidad de extremos que tenga la función.

Otra importante conclusión es que no siempre agregar más neuronas, le agrega precisión a la red neuronal. Pasado cierto punto hace empeorar drásticamente la *performance* de la red.

Otro factor importante a tener en cuenta, es la elección de un patrón de entrada que sea representativo del problema, ya que si no lo es, la red no va a poder generalizar el problema.

La función de activación elegida puede tomar un rol fundamental, dado que al poner como función de activación la exponencial, no pudimos conseguir una buena aproximación.

Uno de los principales problemas que tuvimos, fue el aprendizaje de los extremos del intervalo, dado que al no poder conseguir valores que están en el contorno (porque están fuera del intervalo), no se logró hacer que aproxime correctamente. Este error persistió inclusive, cuando le dimos a la red un patrón de entrada con una gran cantidad de valores en los extremos.

A. Anexo

A.1.1 Tabla de errores de los tamaños de *input* con 100 iteraciones

Neuronas	Input Pesado	Input Reducido	Input Uniforme
1	3.083743	1.028420	1.007395
2	0.418185	0.611843	0.306380
3	0.010840	0.035773	0.014258
4	0.023786	0.033954	0.014170
5	0.017633	0.044339	0.014606
6	0.018374	0.032133	0.022124
7	0.014663	0.038875	0.012999
8	0.012126	0.045801	0.012560
9	0.023151	0.034368	0.018116
10	0.048659	0.030895	0.022754

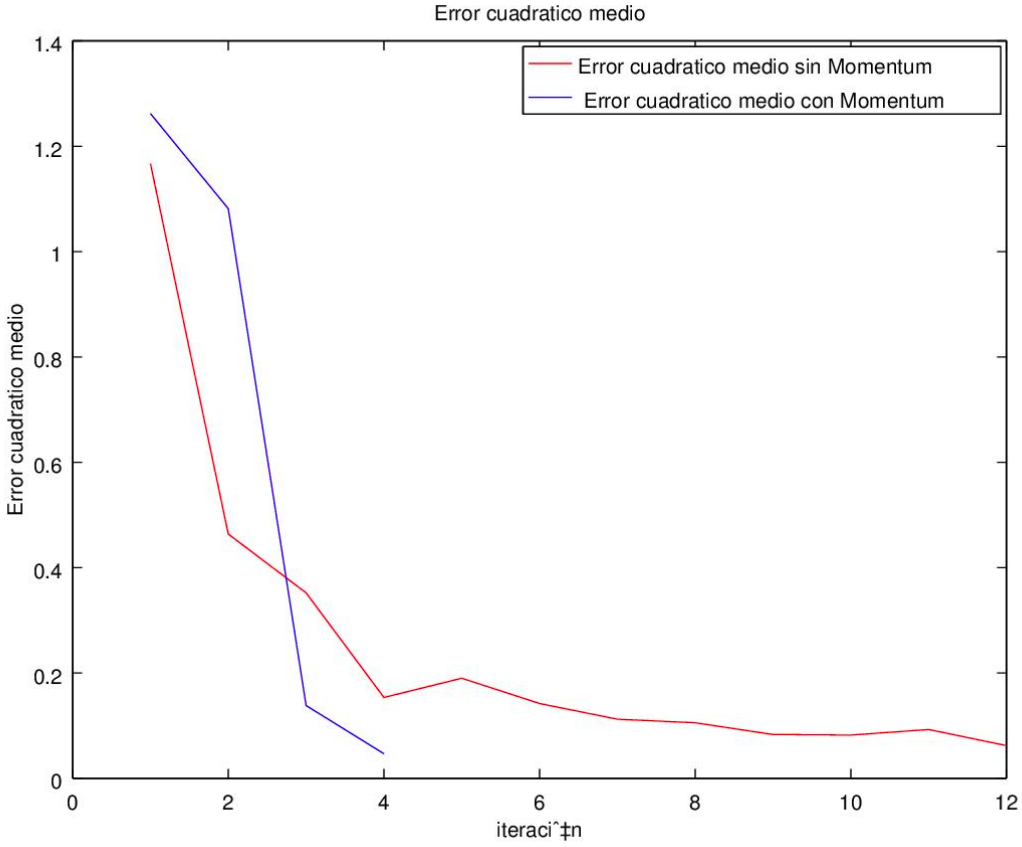
A.1.2 Tabla de errores de los tamaños de *input* con 1000 iteraciones

Neuronas	Input Pesado	Input Reducido	Input Uniforme
1	2.633577	0.879829	0.944424
2	0.191018	0.239681	1.111854
3	0.013454	0.062525	1.758705
4	0.011560	0.028750	0.012690
5	0.011898	0.027034	0.014153
6	0.009929	0.025518	0.013235
7	0.014037	0.024261	0.012661
8	0.014212	0.021872	0.012682
9	0.012102	0.032671	0.015582
10	0.014447	0.026864	0.012481

A.2. Tabla de resultados de parámetros adaptativos (reducida)

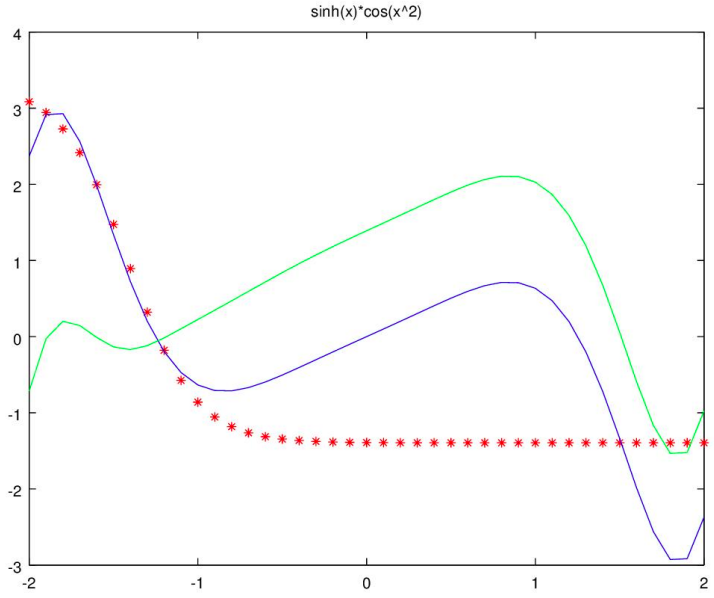
η	a (*etainc*)	b (*etadec*)	Incrementos	Decrementos	Error
0.050000	0.040000	0.100000	15.750000	83.250000	0.010966
0.100000	0.030000	0.200000	33.250000	65.750000	0.011824
0.100000	0.040000	0.100000	16.000000	83.000000	0.012436
0.050000	0.050000	0.200000	24.250000	74.750000	0.012912
0.100000	0.040000	0.200000	24.000000	75.000000	0.013504
0.050000	0.030000	0.200000	30.250000	68.750000	0.013678
0.050000	0.020000	0.100000	35.750000	63.250000	0.013691
0.100000	0.020000	0.100000	29.250000	69.750000	0.013808
0.050000	0.030000	0.100000	26.250000	72.750000	0.015431
0.050000	0.050000	0.300000	22.250000	76.750000	0.016084
0.100000	0.020000	0.200000	34.500000	64.500000	0.017257
0.100000	0.030000	0.100000	16.000000	83.000000	0.017328
0.100000	0.040000	0.300000	25.000000	74.000000	0.017539
0.100000	0.010000	0.100000	35.250000	63.750000	0.017714
0.050000	0.020000	0.300000	15.250000	83.750000	0.018243
0.100000	0.010000	0.200000	19.000000	80.000000	0.018878
0.100000	0.030000	0.400000	14.000000	85.000000	0.018922
0.050000	0.020000	0.200000	28.000000	71.000000	0.020155
0.100000	0.020000	0.300000	21.500000	77.500000	0.020996

A.3. Comparación de errores con y sin *momentum*.

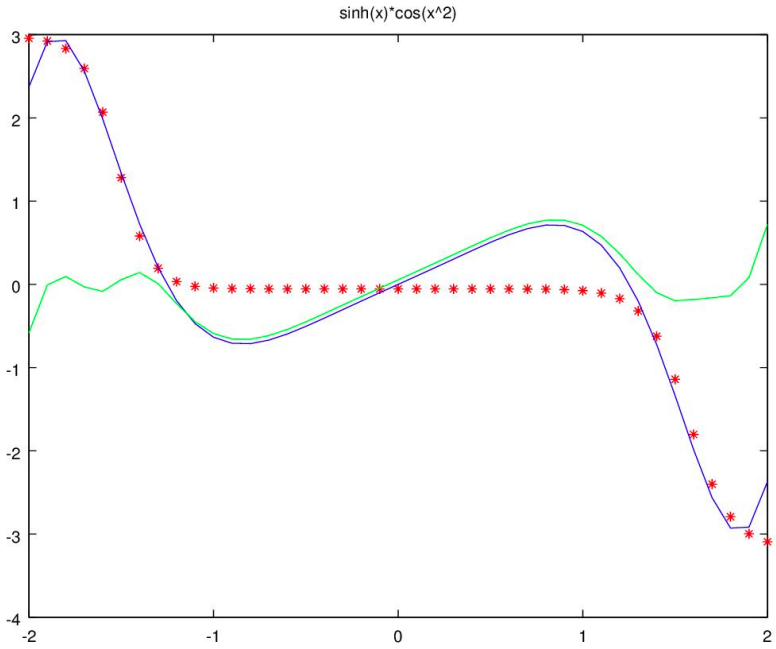


A.4 Variación de cantidad de neuronas

A.4.1. Una neurona



A.4.2. Dos neuronas



A.4.3 Tres neuronas