TPE - N2

Sistemas de Inteligencia Artificial

Redes Neuronales - Perceptrón multicapa

Integrantes

Alan Pomerantz
Tomás Lori
Santiago Ayuso

1. Introducción

Se realiza una implementación de una red neuronal multicapa con aprendizaje supervisado, utilizando el *Algoritmo de Backpropagation* para corregir iterativamente el peso de las conexiones. Se analiza su comportamiento variando la arquitectura de la red, el valor del factor de aprendizaje y los parámetros de las mejoras del algoritmo desarrolladas. Por último se analizan los resultados, y se los compara, sacando las conclusiones correspondientes.

2. Modelado del problema

2.1 Modelado del perceptrón

Para poder representar los pesos de las conexiones dentro de la red, se modela una matriz de tres (3) dimensiones, donde cada capa de la matriz (es decir, cada submatriz de dimensión dos) representa una capa de conexiones entre la capa *i* de neuronas y la correspondiente capa *i+1*.

En dicha matriz, cada columna representa los pesos de las conexiones de todas las neuronas de la capa *i* hacia una sola de la capa *i+1*. En consecuencia, cada fila de dichas matrices representará las conexiones entre una neurona de la capa *i* con todas las de la capa *i+1*.

Como resultado de esto, dichas submatrices de 2 dimensiones tendrán como tamaño la cantidad de neuronas de la capa con mayor número de neuronas y para las capas más pequeñas, serán rellenadas con valor cero (0) donde corresponda.

Por último. al momento de crear el perceptron, a cada capa (submatriz de 2 dimensiones) se le agrega una fila extra en la parte superior para representar las conexiones entre el umbral de dicha capa y todos los nodos de la capa superior correspondiente. En el anexo, se puede encontrar una representación gráfica del modelado del perceptrón.

2.2 Selección del conjunto de entrenamiento y testeo

A partir de la función asignada a este grupo (Ver gráfico en el Anexo)

$$y = tanh(0{,}1x) + sin(3x), \text{ con } x \in [-4, 4]$$

fig1. Función a analizar

se realiza una evaluación de **800 puntos** dentro del intervalo de la función para obtener tanto el conjunto de entrenamiento, como el de testeo.

2.3 Funciones de activación

Se implementaron las dos funciones de activación requeridas por el enunciado
La función de activación tangente hiperbólica*:*

$$f(x)_1 = tanh(\beta x)$$

fig2. Función activación tangente hiperbólica

y la función de activación exponencial*:*

$$f(x) = \frac{2}{1 + e^{-\beta x}} - 1$$

fig3. Función activación exponencial

La función exponencial fue modificada para que la misma tuviese por imagen al conjunto [-1,1] debido a las características analizadas de la función a estimar.

2.4 Entrenamiento del perceptrón

Con el objetivo de realizar el entrenamiento, se seleccionan los valores de entrada del conjunto de entrenamiento y se calcula su correspondiente salida, utilizando al perceptrón.
Luego, para cada época se calcula el error cometido usando como medida al ECM (Error Cuadrático Medio: *promedio del cuadrado del error cometido por cada valor sobre los cuales se entrenó*). Si el mismo es menor a un valor arbitrario *épsilon* ingresado por parámetro, se define por terminada la etapa de entrenamiento y se retornan los valores de los pesos de las conexiones de la red para proseguir a la etapa de testeo. De no ser así, se aplica iterativamente el *algoritmo de backpropagation* hasta lograr alcanzar el ECM deseado.

Es importante aclarar que los patrones de entrenamiento no se procesan en el mismo orden en cada iteración, sino que se mezclan de manera aleatoria para un mejor entrenamiento de la red.

3. Mejoras sobre la implementación

El algoritmo de backpropagation puede ser optimizado ya que requiere de muchas iteraciones en ocasiones donde se busca estimar en función de un ECM arbitrariamente chico.

3.1 Modificación en ***Regla Delta***

Dentro de la regla del cálculo Delta, y con el objetivo de evitar atascarse en mínimos locales cuando la derivada de la función de activación tiene valor nulo, se redefine la regla delta tal como sigue:

$$\delta_i^\mu = [g'(h_i^\mu) + 0{,}1] * e \quad \text{donde } e = (S_i^\mu - o_i^\mu)$$

fig4. Modificación Regla Delta

De esta forma se logra una solución de compromiso entre el error cuadrático medio y la medida entrópica que mantiene los δ's distintos de cero.

3.2 Moméntum

Dentro del *algoritmo de backpropagation,* el gradiente descendente puede ser muy lento si η es pequeño y puede oscilar si es demasiado grande. Para ello aplicamos una mejora en el descenso promedio, pesándolo y agregándole un nuevo término al cálculo, basado en el momentum o inercia original, de los nuevos valores de los pesos del perceptrón:

$$\Delta w_{pq}(t+1) = -\eta \frac{\partial E}{\partial w_{pq}} + \alpha \Delta w_{pq}(t)$$

fig Moméntum

Para los cálculos, se utilizó un valor de alpha = 0.9.

3.3 Eta adaptativo

Cuando en una cantidad de pasos, determinada por parámetro, el error disminuye, quiere decir que tendremos que aumentar ŋ para no permitir el rápido descenso del mismo. De otra manera si el error aumenta decrementaremos ŋ y volveremos al estado anterior de la matriz de pesos. El crecimiento se realiza a valor constante y el decrecimiento se realiza en un porcentaje de ŋ.

$$\Delta\eta = \begin{cases} +a & si \;\; \Delta E < 0 \;\; consistentemente \\ -b\eta & si \;\; \Delta E > 0 \\ 0 & en \; otro \; caso \end{cases}$$

fig5. Eta adaptativo

[a=0.01, b=0.9. *Consistentemente* está basado en los últimos **10** pasos del algoritmo]

4. Arquitecturas

La arquitectura de la red neuronal tiene una muy importante influencia en el tiempo de aprendizaje, el error cuadrático medio logrado y la capacidad de generalización de la red. No se utilizó una arquitectura de perceptrón simple ya que el problema no es linealmente separable. A modo prueba de concepto, se probó una arquitectura con una sola neurona en la capa oculta, y se la entrenó. Dicha configuración no logra poder aprender el conjunto de entrenamiento y falla al generalizar la función, con el conjunto de testeo como entrada.
En los resultados evaluaremos la utilización de distintas arquitecturas.

5. Resultados y Conclusiones

Para poder analizar los resultados, se plantea el siguiente plan de pruebas.
En una primera instancia, se realiza la comparación entre la utilización y la no utilización de las mejoras y variaciones desarrolladas al algoritmo.
Luego se realiza la comparación entre distintas arquitecturas de red neuronal, variando la cantidad de capas ocultas y la cantidad de neuronas en dichas capas.
Por último, se realiza la comparación fijando los parámetros y alternando únicamente la función de activación, entre la función tangencial y exponencial.
En numerosos casos, se tuvo que modificar el error cuadrático medio para el cual el perceptrón debía finalizar el entrenamiento, ya que de otro modo se requerían muchas iteraciones (traducidas en tiempo) para lograr el valor umbral deseado. Los resultados se encuentran en la sección Anexo I.

5.1 Efecto de las mejoras

Las optimizaciones utilizadas modifican la cantidad de épocas requeridas para obtener las aproximaciones con un ECM determinado. Se puede observar en los resultados del *Anexo* que el uso de estas técnicas redujo en órdenes de magnitud tanto el tiempo como las iteraciones de aprendizaje.
Notamos una gran diferencia a la hora de realizar experimentos con y sin momentum ya que, con la misma desactivada, se requieren más iteraciones para lograr alcanzar el error cuadrático medio deseado.
Por otro lado, valores muy grandes de alpha hacían que el error cuadrático medio osciláse mucho, lo cual al principio no resultó eficiente a la hora de buscar un error bajo, pero cuando esto se combinó con eta adaptativo, los pasos que oscilaban hacia un peor error cuadrático medio fueron descartados y se conservaron los que llevaban a un error más bajo, mejorando así tanto la estimación realizada como el tiempo que se tardó en llegar a dicha estimación.

5.2 Tamaños de los diferentes perceptrones

Otro factor que requirió de mucha experimentación fue encontrar la arquitectura óptima para mejorar nuestras aproximaciones. En general se vió que el agregado de capas no era tan útil como el agregado de perceptrones. Con una capa fue suficiente, a la vez que un número más elevado de neuronas fue más útil a la hora de reducir el error cuadrático medio.
Un problema con el uso de demasiados perceptrones fue que el aumento del tiempo de entrenamiento no se condecia con el aumento en la precisión de las estimaciones. Según se puede observar en las tablas del anexo a partir de cierto punto deja de ser eficiente agregar neuronas a la capa oculta para mejorar las estimaciones, o más bien, comienza a ser muy costoso el entrenamiento de las mismas mientras que la ganancia en las estimaciones se reduce.

Pudimos observar también, que existe un cierto punto a partir del cual la existencia de pocas neuronas no permiten resolver el problema. Las mismas son incapaces de aprender todos los valores necesarios, ya que se encuentran saturadas mucho antes del final de su aprendizaje por no poder retener toda la información que se necesita. Por todo esto tendían a un error cuadrático medio bastante elevado a pesar de darles mucho mayores tiempos de aprendizaje.

La arquitectura con la cual se obtuvieron los mejores resultados fue del tipo [1 N 1], con N entre 10 y 30. Observamos en las pruebas que el agregado de neuronas a partir de ese punto no reduce considerablemente el error mientras que, con N menor a 10, el error cuadrático empeoraba hasta llegar al punto en que la función no era posible de ser aprendida por la red. Definimos la arquitectura candidata como [1 20 1].

5.3 Funciones de activación

Dentro de las pruebas realizadas se encontró el uso de distintas funciones de activación. Se realizaron pruebas utilizando perceptrones de igual arquitectura con distintas funciones de activación y se compararon las estimaciones obtenidas como resultado para evaluar el efecto de usar las distintas funciones.
De los resultados obtenido se pudo concluir que para nuestro problema en particular la calidad de los resultados obtenidos no varía de manera significativa según en uso de las distintas funciones de activación si bien la tangente hiperbólica produjo en la mayoría de los casos resultados marginalmente más precisos.
Pudo ademas concluirse que las modificaciones en los distintos parámetros como el factor de aprendizaje o la arquitectura del perceptrón afectan de igual manera los resultados independientemente de la función de activación utilizada.

Anexo I

Resultados Experimentales

Referencias

entrenamientoPromError = Error promedio set entrenamiento.
maxEntrenamientoError = Máximo error set entrenamiento.
minEntrenamientoError = Mínimo error set entrenamiento.
generalizacionPromError = Error promedio en la generalización de la función, con la red ya entrenada
maxTestError = Máximo error set testeo
minTestError = Mínimo error set testeo.
k = Iteraciones
tiempo = Tiempo transcurrido

Variación en el tamaño del set de entrenamiento para entrenar la red

alpha = 0.9 beta = 0.3 eta = 0.05 epsilon = 0.001 max k = 1000 red = [1 15 1]

	100 samples	500 samples	700 samples
entrenamientoPromError	0.21896	0.090662	**0.026499**
maxEntrenamientoError	0.43716	0.28343	**0.14140**
minEntrenamientoError	0.0018	3.9462e-05	**5.9434e-05**
generalizaciónPromError	0.80424	**0.21765**	0.803252
maxTestError	2.0241	**1.5351**	2.43577
minTestError	0.0058	**0.0028**	0.019579
k	1000	1000	1000
tiempo (segs)	158.187s	631.237s	1102.53s

Variación en el número de neuronas en la capa oculta

alpha = 0.9 beta = 0.3 eta = 0.05 epsilon = 0.001 samples = 600 red = [1 N 1]

5 Neuronas -> [1 5 1]

	max k = 100	max k = 500	max k = 1000
entrenamientoPromError	0.081508	0.041370	0.035635
maxEntrenamientoError	1.22208	1.18219	0.99140
minEntrenamientoError	1.0130e-04	1.0879e-04	1.2902e-04
generalizaciónPromError	1.7073	0.90812	0.84239
maxTestError	1.8335	1.1276	1.1185
minTestError	0.0028024	0.0011633	4.1918e-04
k	100	500	1000
tiempo (segs)	63.4542s	309.423s	617.801s

20 Neuronas -> [1 20 1]

	max k = 100	max k = 500	max k = 1000
entrenamientoPromError	0.037503	0.016015	0.0017634
maxEntrenamientoError	0.29661	0.2324	0.22140
minEntrenamientoError	0.0011156	0.001047	1.0758e-04
generalizaciónPromError	1.0297	1.0046	0.9014
maxTestError	1.8291	1.7351	1.7279
minTestError	0.0028201	0.0028	0.0028000
k	100	500	1000
tiempo (segs)	264.1612s	710.732s	1622.859s

100 Neuronas -> [1 100 1]

	max k = 100	max k = 500	max k = 1000
entrenamientoPromError	0.058367	0.059494	0.067146
maxEntrenamientoError	0.26579	0.22906	0.23177
minEntrenamientoError	5.9917e-05	3.3928e-05	0.0012485
generalizaciónPromError	1.0188	1.0485	1.1120
maxTestError	1.7622	1.7649	1.7918
minTestError	0.0031124	0.0032657	0.0028526
k	100	500	1000
tiempo (segs)	673.9026s	1864.032s	3722.614s

Variación en la arquitectura (Número de capas ocultas) de la red neuronal

alpha = 0.9 beta = 0.3 eta = 0.05 epsilon = 0.001 max k = 500 samples = 600

	[1 20 1]	[1 10 10 1]	[1 5 10 5 1]	[1 5 5 5 5 1]
entrenamientoPromError	**0.069101**	0.14620	0.16781	0.77311
maxEntrenamientoError	**0.26861**	0.40208	0.39798	2.1805
minEntrenamientoError	**4.5029e-04**	0.0018	4.7089e-04	0.096194
generalizaciónPromError	**1.0770**	0.47027	0.79445	1.2001
maxTestError	**1.1926**	1.2963	1.3038	1.7942
minTestError	**0.0028047**	2.0214e-04	2.9593e-04	2.9035e-04
k	**500**	500	500	500
tiempo (segs)	**719.012**	870.636	1062.123	1260.784

Aprendizaje Adaptativo

Variación en el primer coeficiente

alpha = 0.9 beta = 0.3 eta = 0.05 epsilon = 0.01 max k = 500 samples = 500 red = [1 20 1]
consistencia = 10

	No Adaptativo	[0.001, 0.1]	**[0.01, 0.1]**	[0.05, 0.1]	[0.5, 0.1]
entrenamientoPromError	0.169101	0.070846	**0.064895**	0.060411	0.050845
maxEntrenamientoError	0.46861	0.33372	**0.33274**	0.33513	0.33475
minEntrenamientoError	4.5029e-04	1.0488e-05	**4.1291e-04**	5.2206e-04	3.4997e-06
generalizaciónPromError	1.1770	0.83493	**0.77736**	0.89749	1.0359
maxTestError	1.8926	1.5773	**1.5323**	1.5464	1.7303
minTestError	0.0028047	7.7518e-04	**9.1860e-04**	0.0010074	0.0029955
k	500	31	**25**	26	15
tiempo (segs)	319.012	17.8652	**14.9583**	15.3162	8.94027

Variación en el segundo coeficiente

alpha = 0.9 beta = 0.3 eta = 0.05 epsilon = 0.01 max k = 500 samples = 500 red = [1 20 1]
consistencia = 10

	No Adaptativo	[0.01, 0.05]	[0.01, 0.07]	**[0.01, 0.1]**	[0.01, 0.3]
entrenamientoPromError	0.169101	0.078615	0.066646	**0.064895**	0.064719
maxEntrenamientoError	0.46861	0.30558	0.35791	**0.33274**	0.35362
minEntrenamientoError	4.5029e-04	8.1693e-04	1.9908e-04	**4.1291e-04**	6.6398e-04
generalizaciónPromError	1.1770	0.92096	0.98640	**0.77736**	0.90488
maxTestError	1.8926	1.7011	1.6844	**1.5323**	1.5909
minTestError	0.0028047	0.0035729	0.0027571	**9.1860e-04**	9.7627e-05
k	500	32	29	**25**	35
tiempo (segs)	319.012	18.7543	16.9394	**14.9583**	20.5916

Variación en la consistencia

alpha = 0.9 beta = 0.3 eta = 0.05 epsilon = 0.01 max k = 500 samples = 500 red = [1 20 1]
coeficientes = [0.01, 0.1]

	No Adaptativo	**10**	15	20	25
entrenamientoPromError	0.169101	**0.061895**	0.069743	0.063812	0.076742
maxEntrenamientoError	0.46861	**0.30274**	0.32631	0.29093	0.31321
minEntrenamientoError	4.5029e-04	**4.1291e-04**	5.5718e-04	8.9321e-05	1.9932e-04
generalizaciónPromError	1.1770	**0.84736**	0.90130	1.0315	1.2065
maxTestError	1.8926	**1.5723**	1.7190	1.7983	1.8350
minTestError	0.0028047	**9.1860e-04**	0.0029322	0.0028527	0.0028006
k	500	**25**	28	41	45
tiempo (segs)	319.012	**14.9583**	16.4824	24.3669	27.1961

Generalización de la red neuronal

Utilizando arquitectura [1 15 1]

alpha = 0.9 beta = 0.3 eta = 0.05 epsilon = 0.001 samples = 400 red = [1 15 1]

F(y) = tanh(0.1y) + sin(3y)

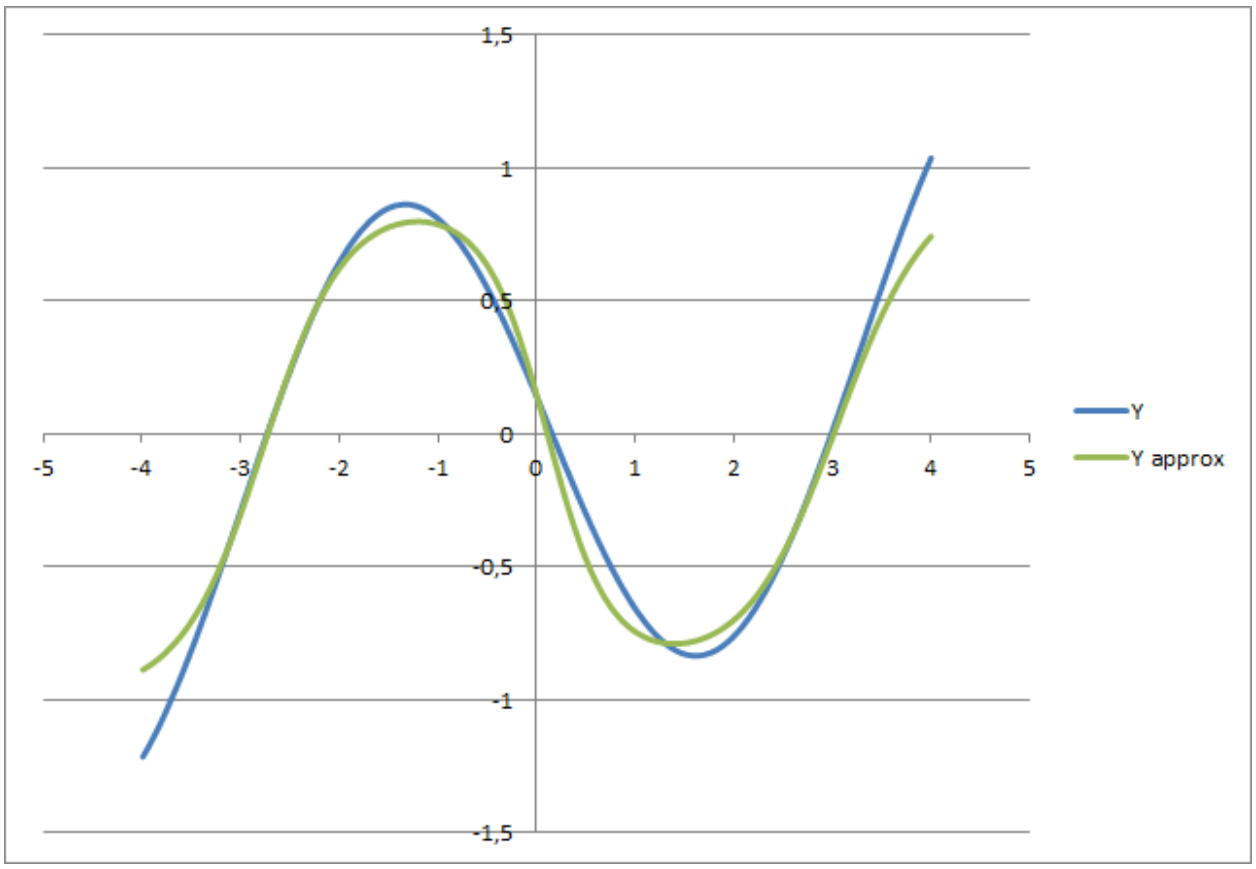

fig6. Generalización de la función

Gráfico de la función

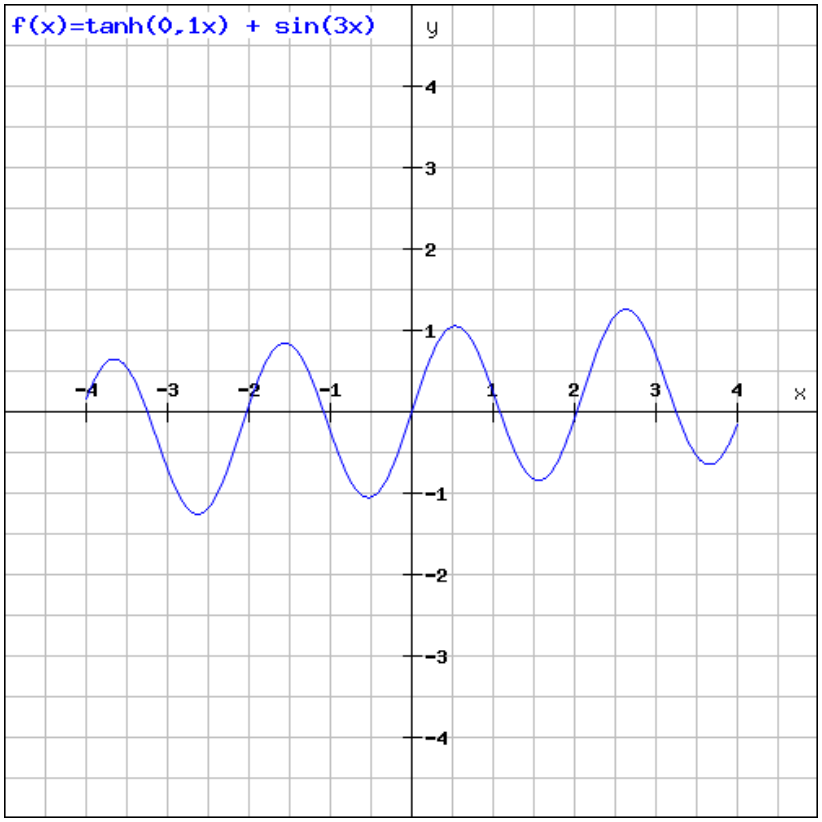

fig.7 Gráfico de la función a analizar

Construcción de un perceptrón

perceptronFactory([x y … z])

Donde [x y … z] es la arquitectura representando la cantidad de neuronas deseadas en cada capa

```
octave:1> perceptronFactory([2 2 1])
ans =

ans(:,:,1) =

   0.66328   0.27098
   0.81538   0.39082
   0.41023   0.73389

ans(:,:,2) =

   0.19398   0.00000
   0.77198   0.00000
   0.84225   0.00000
```

Fig8. Representación de una Arquitectura [2 2 1]

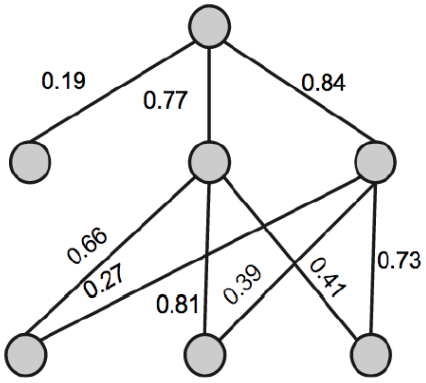

Fig9. Representación gráfica de una Arquitectura [2 2 1]