Métodos de Aprendizaje Automático
Práctico 4

Bruno Garate & Nicolás Izquierdo & Guillermo Siriani

1. Redes Neuronales Artificiales

Las redes neuronales artificales (Artificial Neural Network) son un paradigma de aprendizaje automático inspirado en la biología, es decir, cuentan con elementos que se comportan y organizan de manera similar a las neuronas cerebrales. Este paradigma cuenta con tres fundamentos fuertes:

- Aprende de la experiencia: las redes neuronales modifican su comportamiento respondiendo a las entradas recibidas. Dado un conjunto de entradas, las redes neuronales se ajustan para producir respuestas consistentes.
- Generalizan de ejemplos anteriores a nuevos ejemplos: una vez finalizado el entrenamiento, la respuesta de la red puede ser insensible a pequeñas variaciones en las entradas, lo que las hace especialmente eficaces para el reconocimiento de patrones.
- Abstracción de la esencia de las entradas: algunas RNA son capaces de abstraer información de un conjunto de entradas. Por ejemplo, en el caso de reconocimiento de patrones, una red puede ser entrenada en una secuencia de patrones distorsionados de una letra. Una vez que la red sea entrenada será capaz de producir un resultado correcto ante una entrada distorsionada, lo que significa que ha sido capaz de aprender algo que nunca había visto.

1.1. Implementación

Se implementó la clase ann y la clase Neurona (Si bien puede ser innecesario facilita el entendimiento del código la implementación de esta última clase). La clase Neurona contiene únicamente la función disparar, la cual se encarga de realizar la sumatoria de sus entradas ponderadas y retornandola una vez aplicada la función de transferencia.

La clase ann contiene todas las funciones necesarias para ejecutar el algoritmo, entre las cuales se destacan la función ejecutar, la cual dado una entrada retorna la salida de nuestra única neurona de salida, y la función aplicarBackPropagation, encargada de dado el error de la salida, ajustar los pesos.

En el caso de la función ejecutar simplemente aplica la función disparar de cada neurona perteneciente a la capa oculta, a la entrada recibida. Una vez hecho eso utiliza todas las salidas de la capa oculta como entrada de la neurona salida, utilizando luego la función disparar de esta.
Para aplicar BackPropagation, primero se calcula el error de la salida dado por:

$$E = ValorSalida * (1 - ValorSalida) * (SalidaEsperada - ValorSalida)$$

Luego para cada neurona de la capa oculta x se calcula su error dado por:

$$e(x) = Salida(x) * (1 - Salida(x)) * PesoSalida(x) * E$$

Se ajustan los pesos w_i de cada neurona x pertenecientes a la capa oculta usando:

$$w_i = w_i + factorAprendizaje * e(x) * entrada(x, i)$$

Y finalmente se actualizan los pesos w_i de la neurona de salida:

$$w_i = w_i + factorAprendizaje * E * entrada(i)$$

Se evaluó la aplicación de las funciones sigmoides logística $S(t) = \frac{1}{1+e_{-t}}$ y $tanh$ como funciones de activación de las neuronas. Las ventajas aparentes de $tanh$ sobre la sigmoide son su media nula y las derivadas más suaves cerca de su media, por lo que tiende a saturar menos.

1.2. Entrenamiento

Para el entrenamiento se utilizó la función Sigmoidal y la función Tangente hiperbólica como funciónes de transferencia y una clase auxiliar llamada entrenador lo suficientemente versátil como para poder ingresar la función objetivo, la cantidad de iteraciones y ejemplos, pero además un conjunto de redes neuronales para aplicar el entrenamiento de forma simultanea y realizar comparaciones.

1.3. Pruebas realizadas

x^2

Esta función resulta ser una función sencilla de aproximar mediante una red con pocas neuronas ocultas. Además, demostró tener una convergencia rápida. Esto se debe en parte que es una función convexa con una derivada continua.

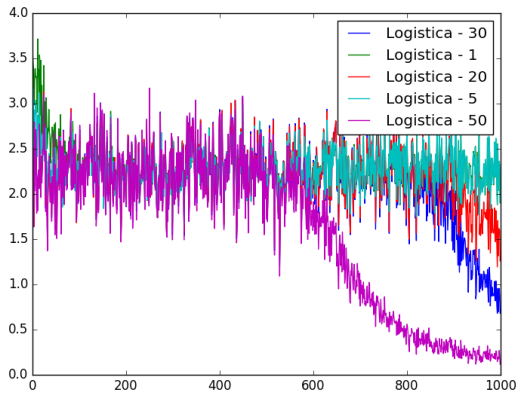

Figura 1: Error a lo largo de 1000 iteraciones al aproximar x^2

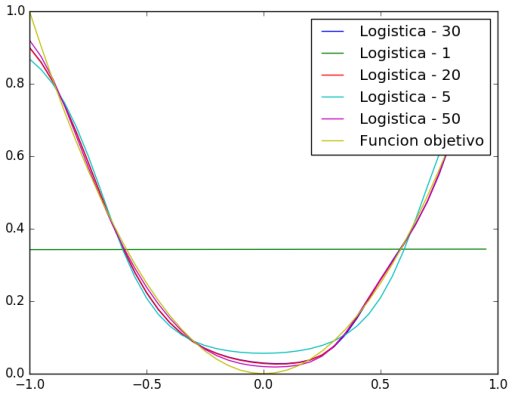

Figura 2: Función x^2 aproximada a lo largo de 100000 iteraciones por redes con 1, 5, 20, 30 y 50 neuronas en la capa oculta

Es interesante notar como en realidad se necesitó un número pequeño de neuronas en la capa oculta para aproximar esta función, pero que a su vez, la convergencia es más rápida cuanto mayor es la red.

$sin(1,5\pi x)$

Para evaluar esta función se debió aplicar un escalado de la salida ya que la salida de la función sigmoidal se encuentra en el rango (0,1). Por eso fue que se aplicó la transformación $T(x) = 0{,}5x + 0{,}5$ de forma de que la función a calcular devuelva valores en el rango [0,1].

Esta se presentó como una de las funciones más difíciles de aproximar mediante la ANN. La función escalada en el eje X se reproduce mucho más fácilmente, probablemente debido a que resulta menos compleja.

Se probaron diferentes combinaciones en busca del número de iteraciones y neuronas en la capa oculta necesarias para aproximar apropiadamente esta función.

Como se puede observar, existe un número mínimo de neuronas necesarias para aproximar esta función. De hecho, cuando el número de neuronas queda por debajo de cierto umbral, la aproximación alcanzada tiende a aproximar una constante 0, que queda resulta ser la línea recta que minimiza la función de error.

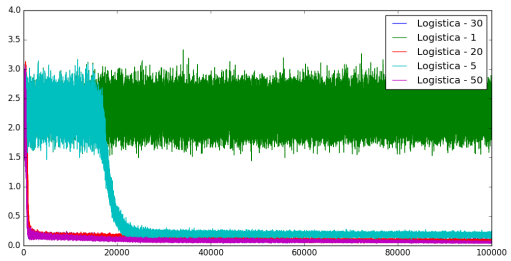

Figura 3: Función x^2 aproximada luego de 100000 iteraciones

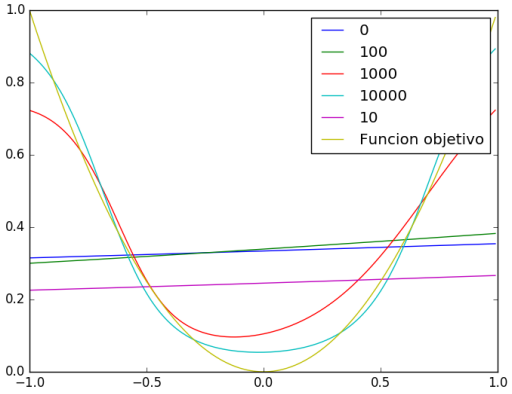

Figura 4: Función x^2 aproximada a lo largo de distinta cantidad de iteraciones

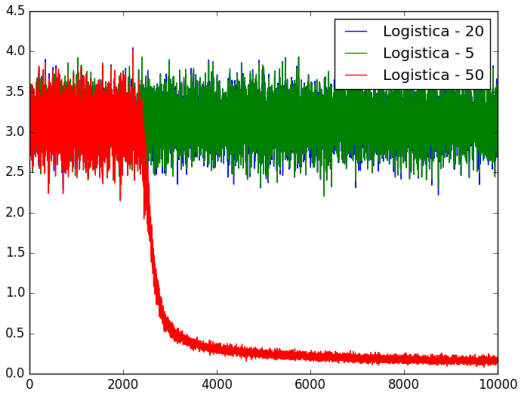

Figura 5: Error a lo largo de 10000 iteraciones al aproximar $sin(1,5\pi x)$

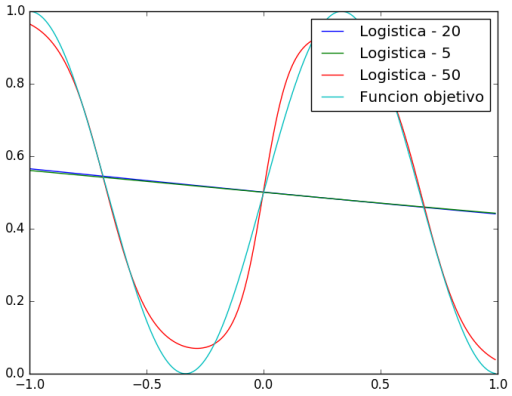

Figura 6: Función $sin(1,5\pi x)$ aproximada por redes con 5, 20 y 50 neuronas en la capa oculta

$$\begin{cases} 1 & x^2 + y^2 < 0{,}5 \\ 0 & sino \end{cases}$$

Además de ser emplearse una dupla para representar la entrada a la red, una de las características a tomar en cuenta de esta función es que no es continua. Esto hace que la función sea imposible de aproximar totalmente. Cabe notar que las derivadas parciales de la función, salvo en la circunferencia de discontinuidad, son constantes. Como vemos en la representación tridimensional de la función aproximada, la aproximación es muy buena en todos los puntos salvo en la discontinuidad.

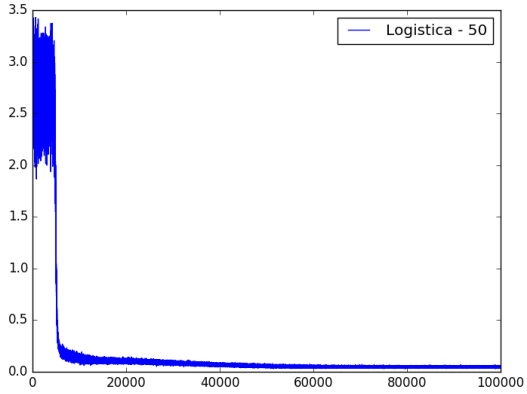

Figura 7: Función $sin(1, 5\pi x)$ aproximada por redes con 5, 20 y 50 neuronas en la capa oculta

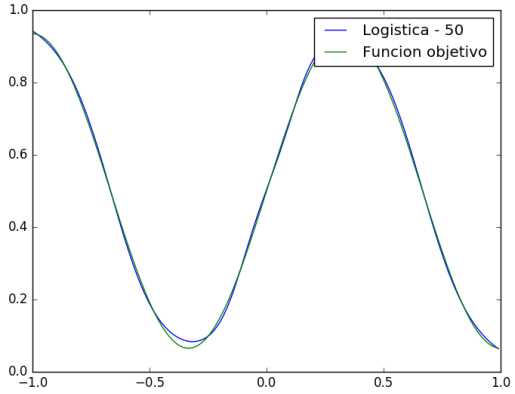

Figura 8: Función $sin(1, 5\pi x)$ aproximada por redes con 5, 20 y 50 neuronas en la capa oculta

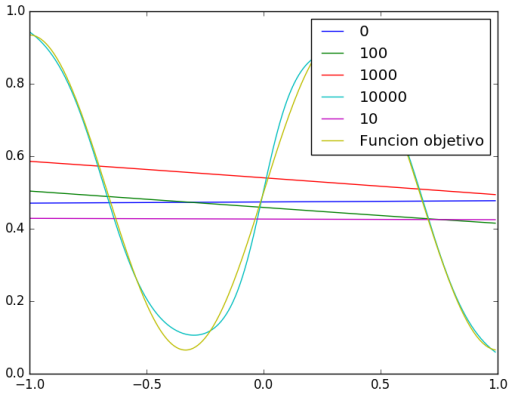

Figura 9: Función $sin(1, 5\pi x)$ aproximada a lo largo de distinta cantidad de iteraciones

1.4. Conclusiones

En este caso fuimos capaces de escalar la función objetivo $sin(1, 5\pi x)$ de modo que pueda ser modelada con la función sigmoide, pero esto no siempre es posible, por lo que la función de *squashing* utilizada debe ser seleccionada de modo que pueda representar todos los valores de la función objetivo. Otra alternativa posible habría sido modelar una red con dos salidas, el valor absoluto del resultado y su signo.

Cuanto mas neuronas se utilizen en la capa oculta, mejor se podrá representar la función objetivo, pero requerirá de mas iteraciones de entrenamiento. Cada función debe ser estudiada individualmente, ya que tendrán distinta cantidad de neuronas mínimas necesarias y distinta velocidad de convergencia, por lo que no existe una única cantidad *mágica* de neuronas e iteraciones que sea óptima para cualquier función. En estos casos es importante destacar que el overfitting no representaba un problema, sino que la minimización de la función error era el único objetivo del entrenamiento. Pero en casos prácticos, muchas veces se deben balancear otro número de factores.

Además las redes parecen presentar una curva característica en el aprendizaje, donde se dan un cierto número de iteraciones donde la red aprende muy lentamente, luego alcanza una fase en la cual el error disminuye rápidamente, hasta alcanzar una segunda fase donde este vuelve a

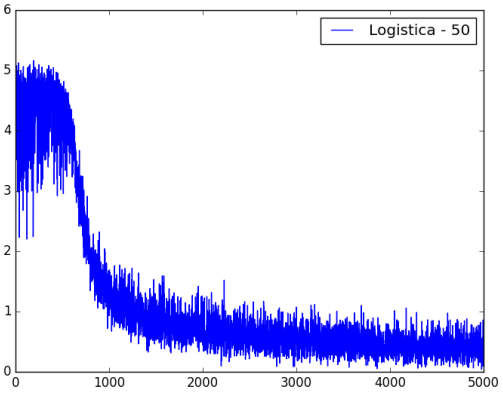

Figura 10: Error al aproximar la función $h(x, y)$ a lo largo de 5000

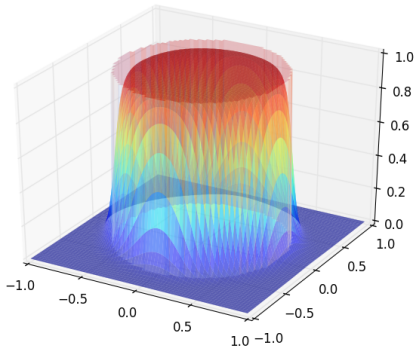

Figura 11: Función $h(x, y)$ aproximada con 5000 iteraciones y función objetivo

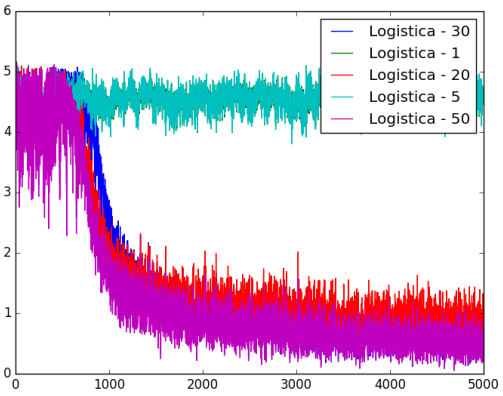

Figura 12: Error al aproximar la función $h(x, y)$ a lo largo de 5000 iteraciones, con redes neurales con 1, 5, 10, 20, 30 y 50 neuronas en la capa oculta

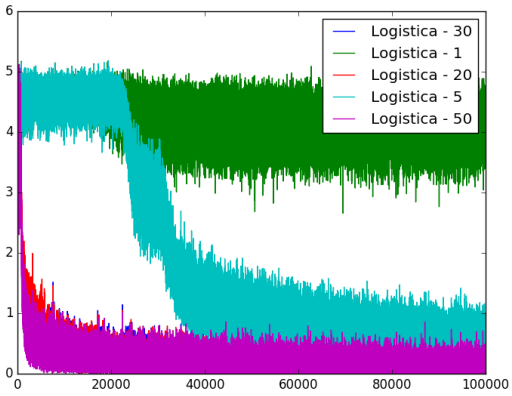

Figura 13: Error al aproximar la función $h(x, y)$ a lo largo de 100000 iteraciones, con redes neurales con 1, 5, 10, 20, 30 y 50 neuronas en la capa oculta

converger lentamente hacía un error "límite". Es posible también que por características de la red, de los datos o formas de entrenamiento o sencillamente azar, la red puede converger a un optimo local o sencillamente diverger. Incluso a veces las redes pueden presentar puntos donde quedan inutilizadas, ya que los pesos de sus neuronas convergen a 0.

La selección de las características apropiadas de la red, sean carácterísticas topologícas, propiedades de las neuronas, la selección de las entradas y las salidas y las dinámica del entrenamiento deben ser seleccionada específicamente en base a las características del caso de estudio.

En tanto a la selección de las redes, además de la *feedforward neural net* empleada a lo largo del informe, también existen otras configuraciones estudiadas. Entre estas se encuentra las *Self organizing maps*, redes neurales recurrentes, Spiking neural networks, Hierarchical temporal memory, Holographic associative memory. Hay redes que especializadas para analizar funciones en dominios espaciales o temporales.

Las *feedforward neural net* son uno de los modelos más simples de redes neuronales. El teorema de aproximación universal que una red neural *feedforward* con una sola capa oculta conteniendo un número finito de neuronas es capaz de aproximar arbitrariamente cerca una función continua en

un conjunto compacto de $\mathbb{R}^n$ bajo ciertas hipotesís sobre la función de activación.

De todas formas, este teorema nos habla de la capacidad de aproximar una función conocida, suponiendo una cantidad ilimitada de muestras sin ruido. En cambio, una característica muy buscada en las redes neurales es su capacidad de generalización. El problema de entrenar una red con una cantidad finita de ejemplos, con cierto ruido, y logrando que esta generalice requiere muchas veces un número de técnicas adicionales. Por eso, es que, aún cuando una red *feedforward* es capaz de aproximar una función con una única capa oculta, se emplean a veces hasta tres (a veces más) capas ocultas.

2. Aprendizaje por Refuerzo

El objetivo en aprendizaje por refuerzo es aprender una función que vaya de *estados* a *acciones* y que permita maximizar una función de ganancia. En este caso el mundo es una matriz de *casillas* y las acciones son las direcciones en las que me puedo mover. Se busca aprender el mejor camino desde cualquier casilla inicial hasta la meta, teniendo en cuenta que hay algunas acciones en determinadas casillas que otorgan cierta ganancia (que puede ser positiva o negativa). Para esto se utilizará el método de Aprendizaje Q (Q-Learning).

2.1. Implementación

Para representar el mundo a ser explorado se implementaron clases que representan Casillas (con sus posibles movimientos y el valor de cada uno), Mundo (matriz de Casillas) y Movimiento (par de casillas, con dirección de movimiento y valor del mismo).

Por otro lado se implementó la clase AprendizajeQ que se inicializa con un mundo a explorar y un factor de descuento. El algoritmo de entrenamiento parte una representación del mundo (que inicialmente es un mundo con las mismas dimensiones que el que se quiere explorar con valor 0 en todos los movimientos posibles de las casillas) y lo modifica de acuerdo a los recorridos realizados. Cada iteración de entrenamiento consta de explorar un recorrido y luego corregir los valores calculados.

En la etapa de exploración se parte de un casilla al azar (corroborando que la casilla no sea la meta) y para cada casilla se elige una dirección al azar entre las posibles para moverse. Esta etapa termina al alcanzar una casilla meta y se retorna una lista con los movimientos realizados.

Una vez obtenido el recorrido de la iteración, se corrigen los valores de las casillas recorridas desde la meta hacia atrás.

2.2. Entrenamiento

El entrenamiento fue realizado para factores de descuento 0.8 y 0.4 partiendo de mundos con casillas con valor 0 en todas las direcciones posibles e imprimiendo el estado del mundo calculado para 5, 10 y 30 iteraciones.

2.3. Resultados

Las gráficas indican la dirección de mayor ganancia para cada una de las casillas. Las casillas sin flechas son aquellas para las cuales todos los movimientos posibles tienen valor 0 en la política aprendida. Se observa que las casillas de la meta no tienen flechas ya que no tienen movimientos posibles en ninguna dirección.

2.3.1. Factor de descuento 0.8

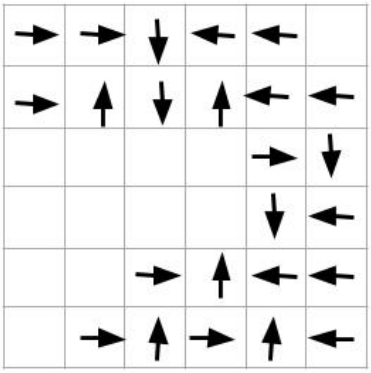

Figura 14: 5 iteraciones

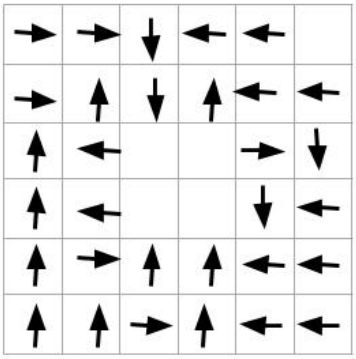

Figura 15: 10 iteraciones

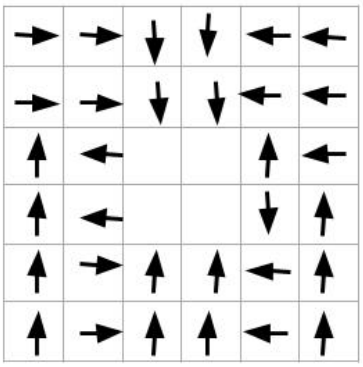

Figura 16: 30 iteraciones

2.3.2. Factor de descuento 0.4

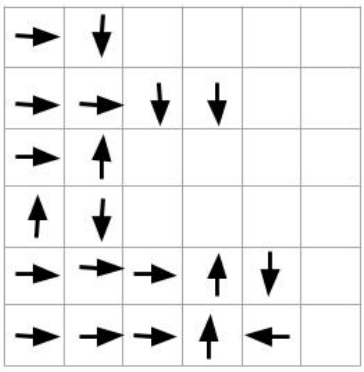

Figura 17: 5 iteraciones

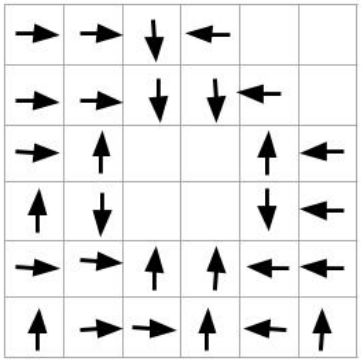

Figura 18: 10 iteraciones

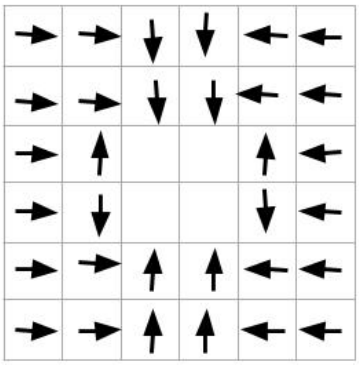

Figura 19: 30 iteraciones

2.4. Conclusiones

Se observa que el factor de descuento seleccionado incide significativamente en cuál será la mejor política a aprender y que cuanto menor sea este, mas importante será encontrar los caminos mas cortos. Además, cuanto mas corto es el recorrido óptimo a descubrir, menos iteraciones se necesitan para aprenderlo, ya que los valores óptimos de ganancia deben transmitirse por menos casillas. En la ejecución mostrada, el algoritmo no fue capaz de aprender una política óptima para el factor de descuento 0,8 (es decir que existen caminos que otorgan mas ganancia que no están siendo considerados) pero si para 0,4.

Quedó pendiente implementar una selección de recorrido de entrenamiento que pondere explorar nuevos caminos y reutilizar información ya conocida, pero por el tamaño del mundo a explorar y teniendo en cuenta los buenos resultados obtenidos consideramos que esto no representaría una mejora significativa, al menos para este caso.

Referencias

[1] Y. LeCun, L. Bottou, G. Orr and K. Muller *Efficient BackProp*, Neural Networks: Tricks of the trade, Springer, 1998

[2] Xavier Glorot, Yoshua Bengio*Understanding the difficulty of training deep feedforward neural networks*, DIRO, Université de Montréal, Montréal, Québec, Canada